Development of a tiflo-application to convert musical notes to Braille

Por
Miguel Zhefan Ye Ye
Álvar Julián de Diego López
Álvaro Antón García

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Desarrollo de Videojuegos
Grado en Ingeniería Informática
Facultad de Informática

Dirigido por
María Guijarro Mata-García
Joaquín Recas Piorno

Desarrollo de una tiflo-aplicación para convertir notas musicales a Braille

Madrid, 2020–2021

Development of a tiflo-application to convert musical notes to Braille.

Desarrollo de una tiflo-aplicación para convertir notas musicales a Braille

Memoria que se presenta para el Trabajo de Fin de Grado

MIGUEL ZHEFAN YE YE
ÁLVAR JULIÁN DE DIEGO LÓPEZ
ÁLVARO ANTÓN GARCÍA

Dirigido por

María Guijarro Mata-García
Joaquín Recas Piorno

Facultad de Informática
Universidad Complutense de Madrid

Madrid, 2021

Acknowledgements

First of all, we want to say thank you to our tutors María Guijarro Mata-García and Joaquín Recas Piorno for letting us take part in this beautiful project, which will let ONCE bring the opportunity for blind people to edit and study music in an accessible way. Thank you for always being available for us and for all the help you gave us during the development of the project.

We would like to thank ONCE for proposing this project and making it possible.

Thank you David Peña Quineche for guiding us during the whole development process of the project and for your constant interest in our work and progress.

And last but not least, we would like to express our appreciation to the SW office and thank David Pacios Izquierdo for giving us the template of LaTeX and for his help throughout the whole career. We appreciate it.

Abstract

The percentage of visually impaired people in Spain is around 6.2%, the highest in Europe [1]. Although society is becoming more and more aware of the difficulties that these people suffer, there are still not enough means. Just as technologies are growing every day, accessible technologies should grow at the same pace as well. One of the fields where accessibility is still lacking is music. Although new music software has been developed in recent years, many of them are either still under development or have a high cost.

In order to make music more inclusive, the ONCE organization decided to create the *LiveDots* project. Until now *LiveDots* was an application capable of displaying scores in braille and ink from XML files, allowing the use of a screen reader, specifically *JAWS*, to dictate the score [2].

After the pleasant reception by ONCE, it was decided to further develop the project, expanding the areas of file reading, music playback, and score modification. Always maintaining the same level of accessibility through the program and its proper functioning.

During this year our work has consisted of developing these new functionalities with the purpose of making *LiveDots* continue growing and becoming a tool that offers an accessible option not only for the representation of the music but also for its learning and composition.

After finishing the development, the application was tested by members of ONCE, who work daily in the field of developing accessible applications for blind people, in order to verify that the standards of accessibility were met.

LiveDots is now a tool that takes one more step towards a more inclusive society in the field of technologies. For this reason, the development of the *LiveDots* application will become part of a much larger project by ONCE that will continue to provide blind users with the opportunity to approach the world of music.

The *LiveDots* repository can be viewed at the following link: `https://github.com/alvant01/TFGLiveDots`.

Keywords: Braille, visually impaired, blind, music, inclusive, music player, score, WPF, Manufaktura, C#.

Resumen

El porcentaje de personas con alguna discapacidad visual en España es alrededor de un 6.2%, el más alto de Europa [1]. Aunque la sociedad está cada vez más concienciada con las dificultades que estas personas sufren, aún no hay suficientes medios. Al igual que las tecnologías crecen cada dia más, las tecnologías accesibles deberían hacerlo al mismo ritmo. Uno de los campos en los que aún falta accesibilidad es la música. Aunque en los últimos años nuevos programas musicales han sido desarrollados, muchos de ellos o están en desarrollo o tienen un coste elevado.

Con la finalidad de hacer la música más inclusiva la organización ONCE decidió crear el proyecto *LiveDots*. Hasta ahora *LiveDots* era una aplicación capaz de mostrar partituras en braille y en tinta a partir archivos XML, permitiendo el uso de un revisor de pantalla, en concreto *JAWS*, para ir dictando la partitura [2].

Tras la grata recepción por parte de la ONCE se decidió continuar el proyecto, ampliando las áreas de lectura de archivos, reproducción musical y modificación de la partitura. Manteniendo siempre el mismo nivel de accesibilidad a través del programa y su correcto funcionamiento.

Durante este año nuestro trabajo ha consistido en desarrollar estas nuevas funcionalidades con el propósito de hacer que *LiveDots* siga creciendo para convertirse en una herramienta que ofrezca una opción accesible no solo para la representación de música sino también para su aprendizaje y composición.

Tras acabar el desarrollo la aplicación fue testeada por miembros de la ONCE, los cuales trabajan diariamente en el campo de desarrollo de aplicaciones accesibles para personas invidentes, con el fin de comprobar que se cumplian los estándares de accesibilidad.

LiveDots es actualmente una herramienta que da un paso más hacia una sociedad más inclusiva en el ámbito de las tecnologías. Por esta razón, el desarrollo de la aplicación *LiveDots* va a pasar a formar parte de un proyecto mucho mayor por parte de la ONCE que seguirá brindando la oportunidad de acercarse al mundo de la música a los usuarios invidentes.

El repositorio de *LiveDots* se puede ver en el siguiente enlace: `https://github.com/alvant01/TFGLiveDots`.

Palabras clave: Braille, discapacidad visual, ciegos, música, inclusivo, reproductor de música, partitura, WPF, Manufaktura, C#.

Contents

List of Figures

Chapter 1

Introduction

In the course of history, music has always accompanied mankind. In many societies, it is even a core part of their culture, becoming indispensable and integral to their society. Every type of human activity may be accompanied by music, anything, from games, ceremonies to work or healing. Music is the way to express life through the medium of sound [20, pp. 123-124].

Music has an important role in our day to day life since it is a way of showing and expressing our feelings and emotions. Moreover, it is a universal way of communication, that is inclusive of all the cultures around the world, regardless of time, place or language. Music indeed is a very powerful tool.

Music provides an important aesthetic contribution to the lives of sighted individuals. While instructional programs for these students include music activities, there are usually more recreational than therapeutic. [21, p. 63)].

However, the aesthetics part becomes secondary for the visually impaired, many of their physical problems are improved using music. Music therapy becomes much more relevant and important in their lives as it helps them to develop their psychomotor skills [21, p. 63)].

As Fred [21, p. 63] states, *"visually impaired infants usually lag behind sighted children in motor development. Those who can see reach out to objects, toys and people, enhancing psychomotor coordination. Sightless infants, though, receive no stimuli and do not develop as quickly. Musical toys offer an important means for physical development. The aural stimulus becomes a substitute for the visual. Infants instinctively extend their hands to grasp sound-producing objects, aiding vital psychomotor development."*

"Musical notation is the visual record of something that was heard, imagined musical sound or a set of visual instructions for the performance of music." [22]. Such a record is written down on paper or saved in a digital format. Besides, the process of capturing sounds on paper is very laborious. This later is used as a way of communication for the sighted person. Since the reader can comprehend what is captured on the paper.

Actually, music theory goes back very far in time, in the 6th century BC, a system that constructs a musical scale originating the circle of fifths was created, this is called the Pythagorean tuning. This system's creation is attributed to the philosopher Pythagoras

and it is a system that is still used today [23], as following this musical tuning system makes tuning by ear very easy because the frequency ratios of all intervals are based on the ratio 3:2, known as the "pure" perfect fifth [24].

Musical notation goes back to very ancient times, the earliest form of musical notation is found in Nippur, Babylonia (which is today's Iraq), about 1400 BC. It is written in a cuneiform tablet and it represents instructions for performing music [3]. Byzantine music once included music for court ceremonies, also written down in some sort of notation, a byzantine musical notation representing an hymnal with daily chant can be seen in Figure 1.1.

Figure 1.1: Byzantine music notation [3].

Musical notation is something that was not used frequently until relatively recent times, in a big part of our world, because it was something that was never felt to be needed. Musical notation captures the fugitive reality of musical notes and allows the reader to time travel back to the moment it was written [25]. Such possibility allows us to share music through time and space, now enhanced even more, with the appearance of digital systems.

But what about the people that cannot see or have low vision, for blind people, the most widespread method used to represent music is called musical braille, which is essentially an adaptation of Braille's both reading and writing system [2]. Braille is a tactile system of both reading and writing that is mostly used among blind or visually impaired people. Braille is named after Louis Braille, who presented the idea of a tactile system that could give the possibility of writing and reading fast and efficiently for blind people [26].

Knowing musical braille gives the blind musician access to a whole new world, it is after all a way of communicating through music and comprehending the musical works of others. Not only this but a blind person's perception of music is very different from a sighted person. Even in the learning period, the way the blind or visually impaired student learns music differs a lot from the sighted students [27].

As of now, in the field of education, there is a gap between the blind and sighted students, and of course, this is also applied to the field of music education. And although it is evident that a lot of progress has been made in recent decades, in some countries such as Ireland, dedicated funds for students with disabilities were created by the recommendation of AHEAD; there is still a lot to be done. Young people with visual impairments are fifty percent less likely to get to third-level education [28]. The goal is to give the students with disabilities an appropriate inclusive education where all their needs are met [28]. In the field of music education, an ideal class would be a mix of both blind and sighted students, where both learn music, the blind student follows the class by learning musical braille notation and the sighted student learns the traditional way, while they also get familiar with the conception of traditional ink music.

For the inclusion of the blind students, there are some strategies and tools that are used to help them participate more, these are mediational means. A few examples would be to use enlarged print notation, increase the usage of musical braille as much as they possibly can, or the creation of school bands or orchestras, to involve these students in something larger and motivate them to learn while also building a social connection within the band [29]. It is vital that the student with disabilities has positive feelings towards participating in class, as this boosts furthermore their learning capabilities. This also boosts social interaction within the band, as they have to communicate with each other. Mixing both blind and sighted students allows the blind students to get more comfortable with the exterior, as restricting blind students from physically moving exacerbates their disability even more, and also moving the students with disabilities to get special instruction is not good for them, as it is a socially stigmatizing event [29, pp. 90-93].

Technology has been helping and is to this day helping immensely towards more inclusive education for blind students. Allowing the possibility of creating tools that integrate the musical conception between the sighted and blind people [2]. There are already programs designed to enable visually impaired people to view and edit music in Braille notation. Such programs use pattern recognition applied to musical notation recognition; usually, bringing accessibility technology to blind people is based on computational intelligence methods such as the one mentioned before. Still, the development of software tools for disabled people is far inadequate to necessities [30].

There is also software that focuses more on the hearing part, by singing the name of the notes so that the information is directly conveyed through the ear [31]; this is also a good teaching methodology to accompany braille learning. There even exists software that handles specificities of Braille Music notation and takes into account the core features of existing formats, this application is called BMML, *Braille Music Markup Language*. Its main aim is to increase the accessibility of Braille musical scores [32]. One last example would-be Braitico, a method of literacy developed by ONCE, *Organización Nacional de Ciegos de España*, for children to learn Braille [33].

ONCE also created EDICO, *Editor Científico de la ONCE* [34], this application was promoted by ONCE and developed with the cooperation of Complutense University of Madrid. EDICO is essentially an editor of mathematics, chemistry, and physics, translating scientific language in ink to scientific notation in braille and vice versa in real-time. [2]. Now, the scope of this application was to be able to go the other way too, from musical braille to ink musical score, while also increasing the functionalities within the program.

1.1 Motivation

With the limitless opportunity's internet allows us, the access to music resources is limitless, from tutorials and documents that allow us to learn more about the rules and composition of a score to programs that allow us to play music to compose our own.

Sadly this range of opportunities shrinks when the user suffers from some kind of visual impairment. This limits the number of people that can enjoy the art of music professionally or just for a hobby.

Another problem people with visual impairments suffer is the fact that the few programs that allow the composition of scores in braille tend to focus only on that aspect and not about the inclusiveness of it, only having in mind people that do not have visual impairment, creating the tendency to not be as developed for people with some sort of visual impairments, alienating more and more of this kind of musicians due to being unable to share their creations with the rest of the world.

There are programs out there in the music department that are accessible to people with disabilities like visual impairment, such as *FreeDots* or *Braille Music Editor*, which will be explained more in-depth below. These software programs facilitate communication overall.

The main issue starts in education, most schools or educational institutions are not fully prepared for students with disabilities such as visual impairment, this does not mean that they cannot learn; it is more because most of the professors are not trained for it as it has not been an important topic until these recent years, moreover they do not have the tools to do so effectively either.

These musical programs for people with visual impairment are useful and accelerate the learning process, as of now all of these issues cause the student with visual impairment to not feel included in the class, as the student cannot participate in any activities or even expressing themselves. There is more work that needs to be done to achieve the ultimate goal, which is a totally inclusive education where any student with disabilities can learn as fast as someone who has no disabilities.

For that reason, the Spanish organization for blind people or ONCE decided to commission an application capable of translating scores from ink to braille and vice versa and allowing its user to play and modify the scores. With this, ONCE wants to create a program that allows musicians regarding their condition to share their passion in an easy and fast way, breaking the barriers that visually impaired people suffer and taking the next step to a more connected and inclusive world.

1.2 Objectives

Our project begins with an already functional application. This application was developed a few years back by students from Complutense University. Its function was to read the *MusicXML* files, interpret them and display two scores: the musical one and the braille one. The main intention was to give accessibility to people that have some sort of ocular impairment or visual disability. This application also can reproduce the musical score that is loaded.

1.2.1 Initial Objectives

Our initial objectives were to expand the scope of the application, to give it more functionality. The proposed objectives were the following:

- Translation from a braille score to a normal score in XML format
- Next was the reproduction of each musical note from the Braille score. The user could move through the braille score but was not able to reproduce the equivalent musical note where the cursor was currently at. The initial idea was to locate where the cursor is and do the necessary things to convert what was being read to something understandable for the library in question and be able to reproduce the correct musical note in return.
- And last, live modification of the braille score, so the user could add more musical notes to it, this in turn also would modify the actual musical score.

1.2.2 Extension of the initial objectives

During the development of the application, we encountered difficulties, which needed to be addressed. After some discussion, we ended up expanding our initial objectives:

- Instead of just reproducing one musical note only, it could be better if the user could select a group of notes with the cursor and the application detects that and automatically reproduces all the notes in order of selection, respecting their respective durations and pitches.
- Previously, the reproduction of single notes was on the same tone, but later we decided to expand it and make it sound more accurate to the actual musical score, the octaves that were taken into account coincide with the number of octaves that are present in a regular piano. This way, the user could either choose to reproduce a group of notes or only single notes as before.
- Restructuration of the code: When implementing the translation from XMLBraille to XML the classes grew too big for what we consider acceptable. To solve this problem we implemented the factory methods to create a more clean and readable project structure.
- Adding more functionalities to the already existing solution. During the development of our objectives, we concluded that we needed to implement or change some functionalities of the given to be able to fulfil our objectives. One of these changes was to make the code register and save in XMLBraille the octaves of every single note.

1.3 Document structure

This document is divided into eight different chapters, each chapter will deal with a different topic. This belongs to a subsection of chapter one, where the introduction of the document is located, along with its subsections, which include the motivation and objectives we had. Here in this subsection, we will explain what each chapter's topic consists of briefly.

Next, is chapter two, this chapter gathers all the initial investigation of the state of the art surrounding our main topic of the project, it is divided into another three main sections. The first one relates to the accessibility of technology that the blind or visually impaired people have; the second section is about Braille in correlation with music, how advanced it is, and what applications we have nowadays; and last but not least, the musical applications that exist and are currently used, explaining what technologies they use and if they are accessible or not for the blind user. Everything related to the already existing technology is gathered in this chapter, whether it is or not for the visually impaired user.

Chapter three is about the working methodology we followed for this project, what technologies were used for the development of it and how we organized and managed all the different internal milestones of the project.

In chapter four, we explain everything in detail about our project and the application *LiveDots*. The first section is dedicated to the application itself, here we show how the project is organized internally, its use cases and how it works in general. The following sections are detailed explanations of the new implementations in the application, this section is divided into another three subsections:

- Braille translation back to *MusicXML*.
- Reproduction of the Braille score with the cursor.
- Real-time modification.

This part was the one that took most of our time in the project. There is an appendix that gives detailed information on the class diagram since the one we used in the first section is simpler and easier to understand. The last part of the chapter is dedicated to talking about the removal of unnecessary/unused things in the program and double-checking that the application does not crash with the elements that were removed.

Chapter five is dedicated to the testing of the newly implemented part of the application, what type's of tests were conducted. What type of testing methods were presented and which ones we used in the end.

In chapter six we give a narrative to each of the members that participated in this project, each one gives their contributions and their experience in hindsight.

Chapter seven gives up possible improvements, enhancements, or extensions that could be made in the application to make it more efficient and more functional, and versatile for the blind user.

Chapter eight gives out the conclusions obtained from the project, including both the development and the analysis and writing of this document.

And lastly, all the appendixes and annexes are located after the bibliography,where if they were written by us, it would be named "appendix" or in their own section (annex) otherwise, as they were test reports done outside the project itself.

Chapter 2

State of the Art/Technological Context

2.1 Accessibility to the technology

The main way an electronic device displaces information is in a visual format, this can prove to be challenging to people with visual impairment to impossible for blind people. To solve these challenges new technologies have been developing giving access to people with visual impairments to technology We can separate these improvements into two categories, the ones that affect the way a user interacts with the machine(inputs) and the ones used to obtain the information, the outputs.

2.1.1 Input devices

Input devices are the systems a user can use to interact with the computer.
Depending on the severity of the visual impairment conventional forms of inputs such as a keyboard and a mouse might be impossible to use. For these cases special software or devices had been developed.
Some examples of this kind of devices and software are:

Dictation software

This kind of software allows the user to interact with the computer by voice commands thanks to voice recognition. This helps the user since for people with visual difficulties moving and interacting with the cursor can be hard to find on the screen.

Refreshable braille display

"A refreshable braille display is a piece of computer hardware which has a series of refreshable, or fluid, braille cells on its surface. Most displays contain a single line with anywhere from fourteen to eighty braille cells. Instead of small holes in a piece of paper, each braille dot in these cells is represented by a tiny pin that can be raised or lowered. This allows individuals who are blind to read the information in braille by running their fingers over the refreshable braille cells and then advancing the display to show the next

set of characters." [35] This device allows visually impaired users to write the braille signs thanks to the buttons allocated in the top part of the device. Furthermore, this device also works as an output since thanks to the braille cells the user can read what is displayed on the screen.

Figure 2.1: BrailleLine. [4]

2.1.2 Output devices

Screen readers

These applications utilize text to speech technology to dictate to the user what is shown on the screen. This type of application focuses on the important parts by analyzing the layout and content, giving only the information needed by the user and not overwhelming them. Examples of this kind of applications are:

- *JAWS*: Developed by Freedom Scientific, Job Access With Speech or *JAWS* is a popular screen reader with an ample set of characteristics such as working with Adobe flash player animations, compatibility with other programs by using *JAWS* Scripting Language or JSL, to name a few. [36]
- NVDA: NonVisual Desktop Access is a free and open code software developed by NVAcess. Written in Python the goal of this project is to give a free alternative to visually impaired users, allowing everyone to access screen readers regardless of income. [37]

Magnifiers

"Screen magnification software, sometimes called screen enlargement software, is computer software that enlarges everything on a computer screen. Because the screen image is enlarged, the user only sees part of the screen at a time. " [38] Some examples are:

- ZoomText: Developed by Freedom Scientific can be purchased with *JAWS* in a package called *FUSION*.
- MAGic: Combines magnification features with screen reading.

2.2 Braille musicography

What is a score? A score is a manuscript or a printed form, called so because of the vertical scoring lines that connect with staves (staffs) that are related to each other. A score can contain a solo work or many parts that unite and form an orchestral [39]. When we talk about a musical score, the general first thought is a physical paper with many staffs on it and in between those staffs, many different musical symbols that form a musical composition. But this is something that a blind person cannot interpret and understand. Due to this, a need arises for blind people. Several different musical representation structures make use of tactile elements to make the interpretation fairly easy.

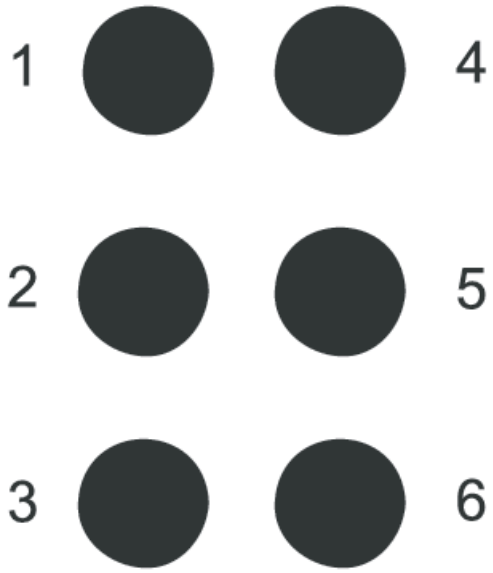


Figure 2.2: Braille 6 dot box [5].

However, the symbols that were created for tactile reading differ a lot from each other all around the world. Braille is also one of those systems, it uses a system of six points (Figure 2.2), they are used nowadays widely in the United States and different English-speaking countries, Louis Braille,the person who invented the braille system already had already taught people how to write in Braille in 1832, and later at around 1860, it was introduced in some of United States' schools [6]. Besides Braille, there have been a lot of different systems. One example is Boston Line Type (Figure 2.3), published around 1834, which used the Roman alphabet without capital letters but it was engraved in something, this way the user could feel all the words that were engraved as they had relief in the

paper, this system would be used after its invention in the United States for over fifty years [6].

Figure 2.3: Boston Line Type [6]

Another example is Moon type (Figure 2.4), which is a new variant of the alphabet in relief, created and still used in the United Kingdom to these days, it is claimed that this system is easier than Braille. One more example would be Fishburne, which is essentially another tactile alphabet, but this one contains more elements than Braille. There are many more examples, all of them have the same objective, to help blind people understand and communicate easily through these systems. As of now, the more widely used and accepted tactile system is Braille. As can be seen in the figure 2.2

Figure 2.4: Moon type [7]

Alternatives emerged as attempts to improve the Braille system, as the way it is, with

only six dots, some words can get very complex and hard to understand. Gabriel Abreu proposed an interesting alternative system as shown in the following Figure 2.2, to improve comprehension and achieve a less complex structure, the idea behind it was to expand the number of dots per box up to eight, in this way, many more combinations could be achieved, resulting in a lesser need to read as many boxes of Braille as before. This new system got some popularity and it is called the *Abreu system*, it even won some awards and got international recognition.

Along with this, another important tactile system was being developed in Spain, it is called the *Llorens system* (Figure 2.5), this one, however, is completely different, from its approach and form, *Llorens system* takes the capital letters and numbers of the Roman alphabet as well as musical symbols and places all of them in relief, it is believed that it was very hard to master. A very good advantage of this system was that the symbols were somewhat easily identifiable for the non-blind person, as it was similar to the alphabet they are used to seeing. These systems were designed with music in mind, as the musical notation is complex and the objective was to soften that complexity, the names of Gabriel Abreu and Pedro Llorens deserve a place of honor in the field of music education for blind people [40].

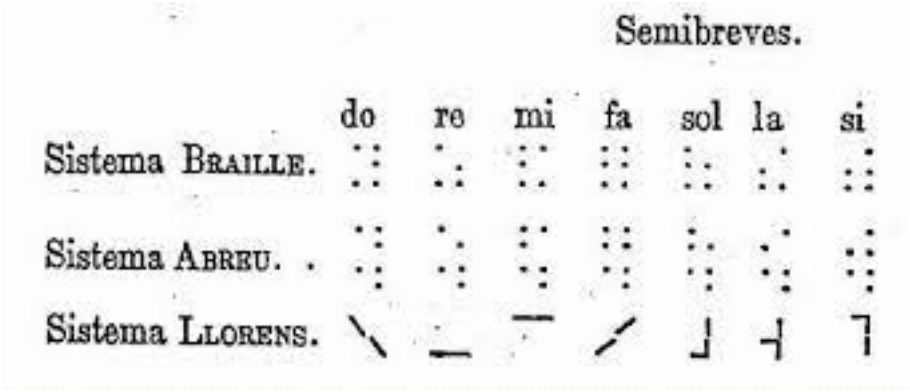


Figure 2.5: Comparison in the representation of seven musical notes in all three types of tactile systems: Braille, Abreu, and Llorens [8].

Although both were fairly used when they were introduced, one of the problems the Braille system had was the limited size of the boxes. In the music field, this would mean that a large number of boxes were needed in order to represent a music score and some ambiguity could happen in the symbols too. However, the *Abreu and Llorens* system was gradually eclipsed still by the system of Louis Braille due to its rising popularity and universalization [40].

Although the Braille system was used universally, many regions around the planet were adapting it with their forms. Besides, Braille was already being modified and adapted depending on the zone it was being utilized in. This would lead to many inconsistencies, especially in the representation of the Braille musical scores in the different countries. Bettye Krolick wrote the *New International Manual Of Braille Music Notation* where it is based on the Braille system of six dots elaborated by Louis Braille [41]. This manual's objective is to gather all the rules needed for the representation of musical Braille, to get standardization and universalization of all the musical scores around the world. This document gathers everything related to music, from the different clefs to the most specific little elements in music. This manual is an enabler to many things, one of them is the

possibility to develop trustworthy software, as this musicography system is something reliable, allowing people from all over the world to work on the same format for better efficiency and communication.

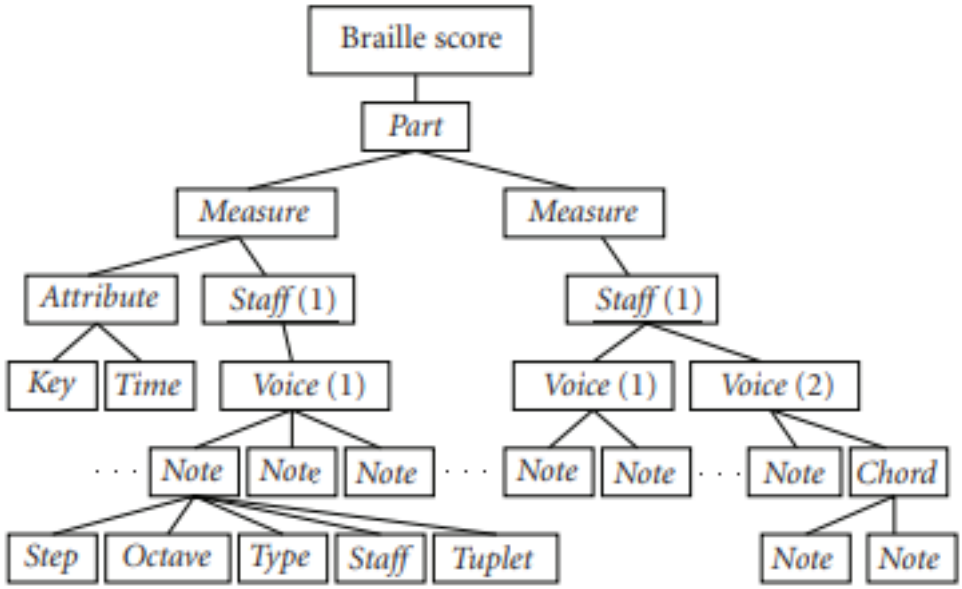


Figure 2.6: A sample tree structure of elements in Braille music notation [2].

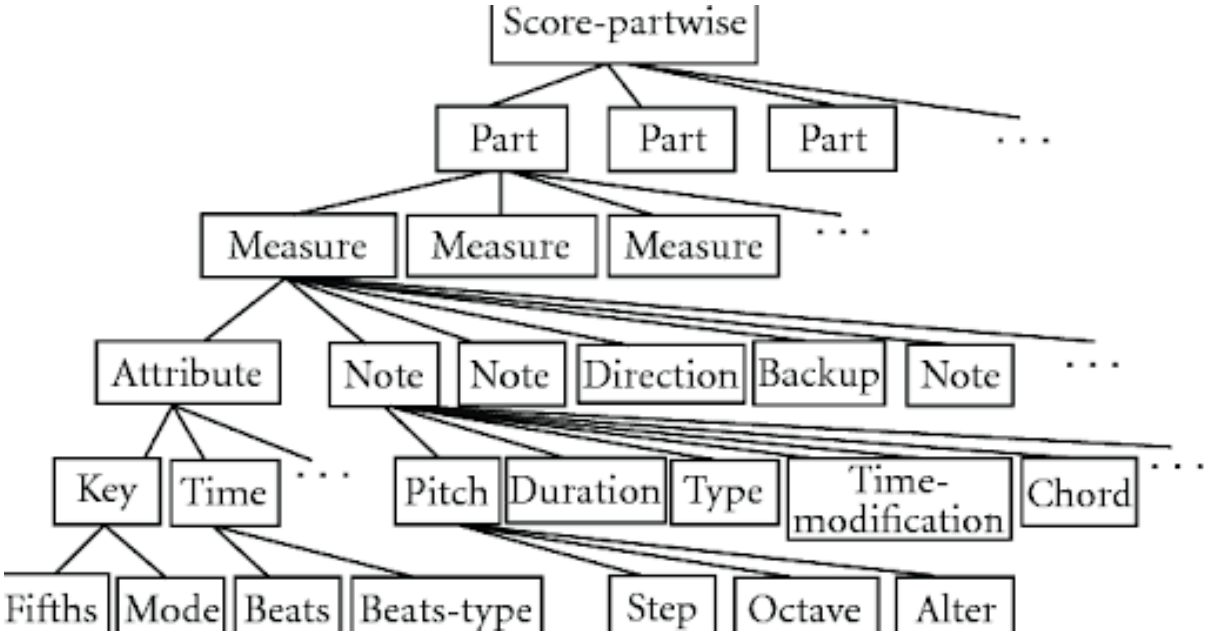


Figure 2.7: A sample tree structure of elements with the *MusicXML* format [9].

Another thing to highlight is *MusicXML*, which was invented by Michael Good, it is an open XML-based file format used to represent western musical notation [42]. There has been a lot of investigation and studies to computerize Braille musical scores, on this line, many programs have been developed that interpret, write, edit or manipulate musical scores. It is worth mentioning investigations such as A Transcription System from

MusicXML Format to *Braille Music Notation* [9], where they show a way of converting a musical score in *MusicXML* format to a Braille musical score. The most interesting part of this project is how they approached the resolution of the conversion, which is by following a tree format, in this way they break and separate everything up and find a proper correspondence for the musical notation between musical Braille (Figure 2.6) and *MusicXML* (Figure 2.7).

2.2.1 Music editors for blind people

Any type of musical application will need to store some sort of musical information, which is usually a musical score and all its elements. For example, a single musical note, which is the smallest element of the score, needs to have its tone and rhythmic duration defined. That is why all these applications where there is some use of musical notation or musical editing need to have some sort of structure to store all this information.

Most of the applications nowadays use *MIDI*, *MusicXML* or both as formats of musical notation. This differs from audio file formats, such as *.mp3* or *.wav*, as these are not designed to be edited specifically, at least at first thought, they are designed to be reproduced. Compared to the first two that were mentioned, which are designed to be edited and modified. Although both are meant to be used as storage from musical files, their end goal is slightly different at the beginning.

MIDI, *Musical Instrument Digital Interface*, was conceived in 1983 to communicate with music synthesizers. Initially, it only communicated two physically separated devices: controllers and sound modules, this can be seen on Figure 2.8. And the *MIDI* connection was unidirectional, the transmitter was usually a keyboard or something piano-like, and then the sound module was an electronic device that is capable of generating sounds of the musical pitch corresponding to the note the musician was playing on the controller [10]. Once the standard of this format was established, it got popular very fast and it is used a lot nowadays in many musical software applications.

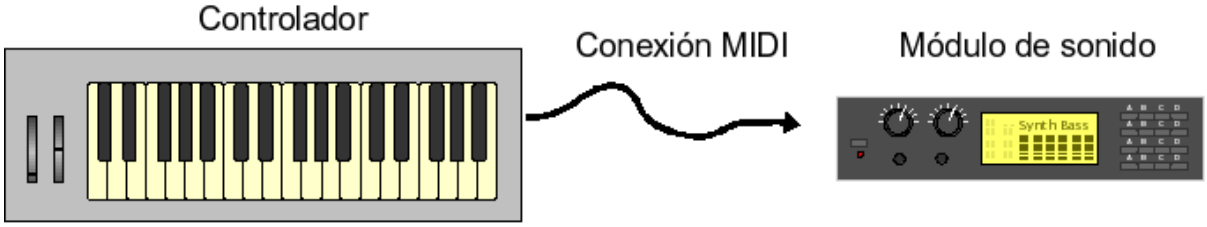


Figure 2.8: Controller on the left connected through *MIDI* to a sound module [10].

On the other hand, there is *MusicXML*, which was published in 2004, have a very big difference between each other, "*in MusicXML format, music is represented by semantic concepts that underlie the musical notation itself*" [43], while *MIDI* consists only of a data string that represents note values in a sequence and their parameters. This results in *MusicXML* being much more accurate in representing the whole musical score. Although more precise, this means that its compatibility with software is slightly reduced, so the advantage of *MIDI* is its high compatibility. Still many modern programs, such as *Finale*, *Guitar Pro*, or *Sibelius* support *MusicXML*.

The applications that are accessible typically support *MusicXML* and make use of it as their musical notation format, this opens up the possibility of importing and exporting

musical scores outside the application itself, increasing the utility of it. A few examples of already existing applications, which are the following: *Braille Music Editor*, *Dancing Dots* and *FreeDots*. Its functionalities will be explained below.

Braille Music Editor

A very popular and interesting application is *Braille Music Editor*, acronym BME, shown in Figure 2.9. This application's origins are in Italy, it is a paid program that is essentially a tool that allows blind musicians to interact with music scores [44]. This includes any type of interaction: write music scores, check them, modify them and even print them out. The music writing follows the rules defined in the *New International Manual Of Braille Music*. This application is very versatile since it allows the user to edit musical scores of multiple tracks. The musical score the user is editing can be checked in many different ways: through an output on the screen-reader that pronounces all the musical elements (it supports *JAWS* and *NVDA*), through the *MIDI* sound, or directly onto the Braille display. This musical score can later be reproduced with different instruments and/or exported as a *MusicXML* file or *MIDI* file [11].

This *MusicXML* file can later be visualized on a lot of traditional music programs such as *Finale*, which we mentioned above. Braille Music Editor also accepts files generated outside its environment. However, the application has its own standard file format, which is called *braille music markup language* (*.bmml*). This file type is actually an extension of *MusicXML* but also supports Braille musical scores. Currently, this application is in its second version (BME 2.4) and it is still being developed.The application was launched more than twenty years ago(2000) and the development is being paid mainly through donations, as these types of applications do not have a large number of users due to their conditioning.

The application consists of a window (Figure 2.9), where the main text is written in Braille format. The user can directly edit the file by typing characters in Braille. In case any musical file is loaded (*MusicXML*), then it would be translated to Braille and displayed on the application's window, this one can also be edited once it is finished loading. Once the file of the music score is finished being edited by the user, it can be reproduced, either totally or partially and its configuration can also be modified. This musical score can be reproduced by many different instruments, all included within the range of standard *MIDI* instrument sounds. For an application to be accessible, it needs to have many different properties, below this we list the ones that Braille Music Editor has:

a). A screen reader, as mentioned above, *JAWS* or *NVDA*, allows the application to read the braille musical score, the application uses a script that allows it to check and identify all the characteristics of all the musical notes accurately. The voice synthesizer has a few different options to reproduce sound. The screen reader can speak the Braille dot combination the cursor is currently at (Braille box), or it can say the musical element of the current position, in other words, where the cursor is at. This is tricky, as some musical elements require more than one Braille dot combination to be represented (they need more than one box or more than one character), so if the writer is currently trying to reproduce each character that is being written, it will be impossible to tell whether it is or not a musical element, so in this case, the screen reader will always speak the braille dot combination. In

Figure 2.9: Braille Music Editor interface [11].

case the whole braille musical score is fully written and finished, the synthesizer will automatically be able to read and say all the elements as musical characters in its monologue unless the user explicitly asks for a braille box combination read.

b). The application allows the output of the text by braille line, this enhances the comprehension of the braille musical score that was loaded or written in the application, as this gives the user the possibility to not only listen to the musical elements but also read them through the braille line.

c). Although at first is obvious, it is very important to keep in mind that blind people do not have the same capability of using a mouse as a sighted person, as they cannot reach all the options with just that, so it is key to allow them to navigate through all the different options with the keyboard, giving the blind user the possibility to use all the features the application has with just the use of the keyboard. This is also aided by the screen reader, as it tells the user where the cursor is exactly located within the application.

d). The *Braille Music Editor* allows the user to print out the currently opened file if any printer is available and gives the user also some freedom in the configuration of the file about to be printed.

e). The input in the application is with a 6-dot Braille format, the user will have six key-binds that each one belongs to one position in the braille combination. In this

way, for example, if the user wants to write a braille combination with position's two and five marked, the user will have to press the second and fifth key-bind simultaneously.

Nevertheless, the need of having an application like *LiveDots* arises because these applications are not finished being developed, with an unknown date to finish it. *LiveDots* is supported by ONCE, which means that it is handled by an organization, in other words, the project is being produced by them so development times and everything is set from the start and there are requirements to be met. Also, BME is a paid tool, which *LiveDots* isn't, giving the project another advantage. And lastly, our application allows the inverse translation from Braille to ink score, so now the communication between blind and non-blind musicians should be as easy as ever.

Dancing dots

Dancing Dots is an online platform that assists blind people or people with low vision by offering them technology, educational resources, and training. *Dancing Dots* fosters inclusion and accessibility and independence for the visually impaired user [45]. It is also oriented towards professors and it has programs that are oriented towards making music more accessible.

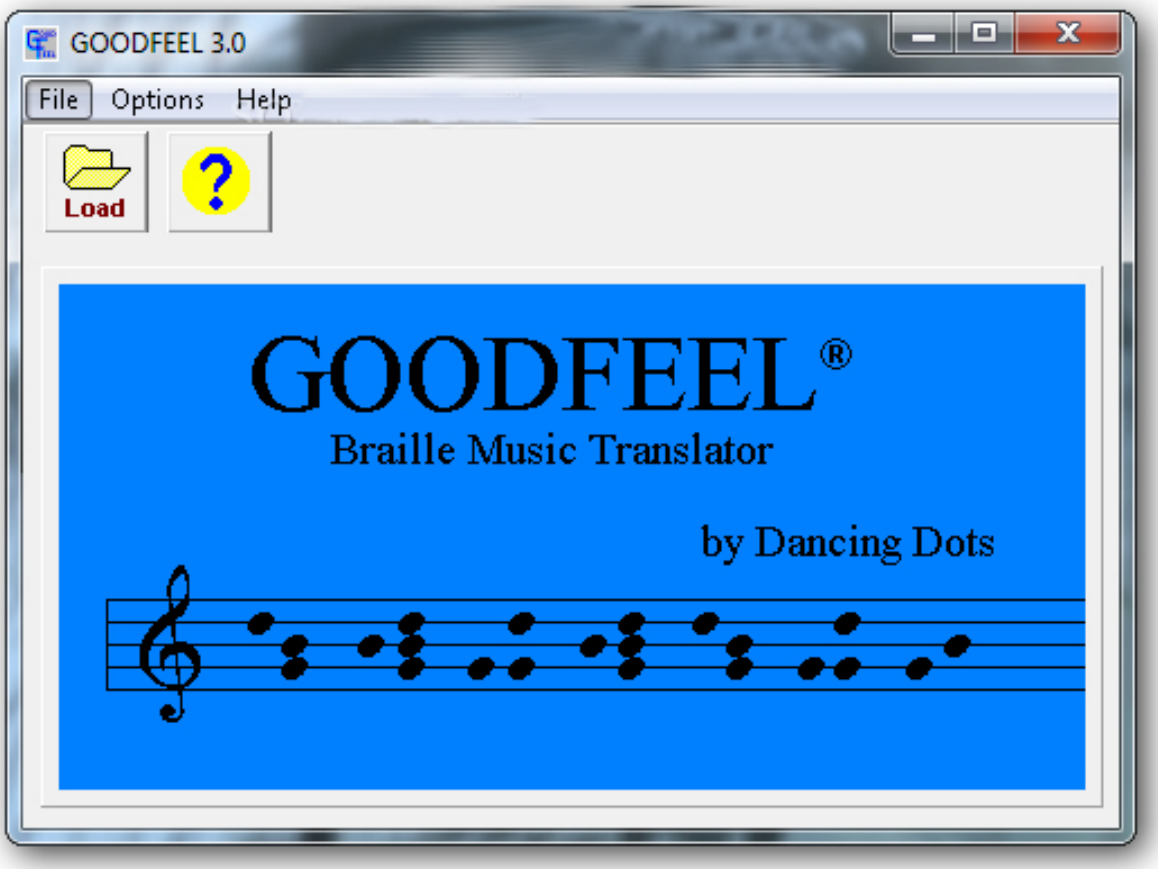


Figure 2.10: GOODFEEL Application [12].

One very interesting program is *GOODFEEL* [46] (Figure 2.10). It is a premium program with a few days of trial. This program allows the user to scan an ink musical score and

convert it to a Braille musical score, accurately and automatically. Blind users can review the Braille musical score with optional verbal and musical sounds and also allow them to edit the score. This application allows the *MusicXML* files for translation to Braille afterward. The screen-readers that are used are also *JAWS* or *NVDA*, just as BME. It is also compatible with many versions of *Windows*, going back to *Windows XP*. It also is compatible with some of the programs the platform offers, such as *Lime Aloud* and *Lime Lighter*.

Another program is *Lime Lighter* (Figure 2.11), which is software designed for low vision people so they can read musical scores. It allows users to read print music with clarity and ease. *Lime Lighter* gives the user the choice of zoom, magnifying the size of the musical notation (up to ten times bigger than the original) while also contrasting it with the background color (also customizable), to increase the visibility of it. It has also a built-in OCR (SharpEye 2, which is also one of the offered programs on their website, it can convert scanned files into *MIDI*, *NIFF*, and *MusicXML* and has a playback feature, this program is also paid with no free trial), allowing the user to scan musical scores. It uses a pedal to avoid using hands in manual scrolling through the musical notes, by pressing the pedal the application will bring up the next bar of music into view; although it can also be configured so that the program goes through the entire musical score with a speed set by the user [13]. This application is also premium but it has a free trial.

Figure 2.11: Lime Lighter being used in a tablet [13].

Lime Aloud is designed for blind people, it works together with *JAWS* for the screen reading part. This application is tied with *Lime*, which is a music notation editor.

Having *Lime Aloud* gives completely blind users the option to use *Lime*; blind users can create music pieces using this application.

Lime, as mentioned above, is a music notation editor that was developed by the University of Illinois [47]. Users in *Lime* can create a score by the manipulation of graphical musical symbols. This application can be paired with other programs the platform offers; for example, using *Lime* paired with *GOODFEEL*, any sighted user can create and prepare a Braille Musical score without any knowledge of musical Braille. *Lime* used to be sold for a price, now it is open to the public, but any donations are welcomed for downloading the software.

Besides all of the programs mentioned here, *Dancing Dots* also has courses in Braille music and offers other products and services, such as podcasts and training online. *Dancing Dots* has many interesting elements that help towards the accessibility of blind or visually impaired people, while they also contribute to the learning sector.

This platform, although very interesting and useful, is not free of charge, and needs multiple programs to work properly for a blind user, while *LiveDots* is an all in once application, giving the user access to all the functions instantly. Hence the need of developing an all inclusive program that is easily accessible for everyone.

FreeDots

FreeDots is an open-source software application that accepts a *MusicXML* file, loads it, displays, and returns a file with the braille translation (braille musical score) (Figure 2.12). This application can be found on the *Git* platform [48]. The application also allows users to reproduce the current loaded musical score. *FreeDots* can also be found in the Google Code Archive which was turned down in 2016, but the application can still be downloaded from there [49].

Figure 2.12: FreeDots application screenshot.

Later on, inspired by this application, Nicholas Foment tried to upgrade the usage of this application by creating a website that would do the same function as the application *FreeDots* but with no need of downloading nor installing any software in the user's computer whatsoever. However, the application itself is run internally by the same software as *FreeDots* [50].

2.3 Other musical applications

Until now *LiveDots* was an application that could open *MusicXML* files and show the notes both in ink and braille notation. But since one of the main goals of this project is to achieve a system for editing the musical score, studying how the most important score editors work is a vital priority.

We are going to research the top available score editors in the market and what specific aspects we can learn from them to make our project more professional and useful.

2.3.1 Finale

Finale is a musical notation software that allows the composer to compose music in a dynamic way to make music projects easier and more manageable [14]. It can manage any type of modifications on the score and has multiple available instrument libraries. It was created by the company *MakeMusic* and it is sold for 600$.

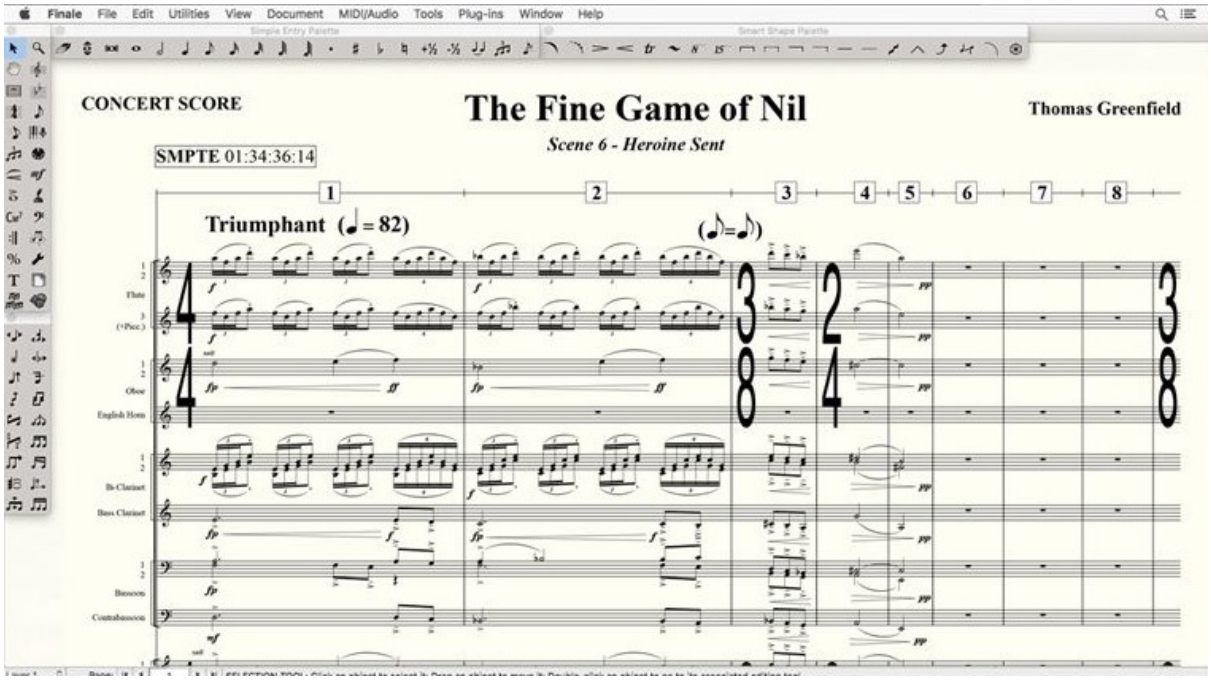


Figure 2.13: Finale Software [14]

Finale supports keyboard and mouse input as well as *MIDI* input, which is important for professional musicians.

It has a variety of different styles and options to print (Figure 2.13) the produced scores and can export the scores into multiple formats such as:

- MP3 audio
- WAV
- AIFF
- MIDI
- PDF
- MusicXML

This software is interesting for us because it is what real professionals use to compose their music and also because it exports scores into *MusicXml* files, just like *LiveDots* does. However, even if *Finale* brings many useful and top functionalities, it is by no means what can be called accessible software. *Finale* does not take into account those users whose vision loss prevents them from seeing screen content or navigating with a mouse.

2.3.2 Sibelius

Sibelius is a very competent musical notation software [15] that allows users to create scores for any instrument and style. This software was created by the company *Avid*.

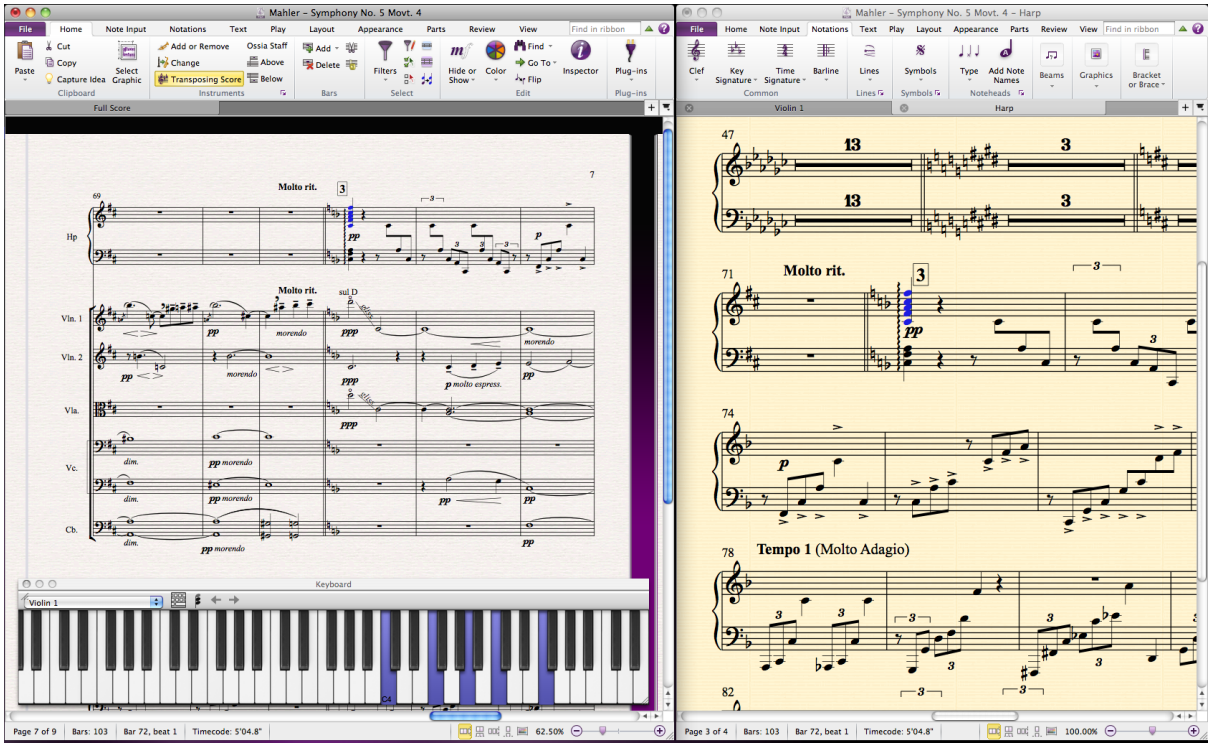


Figure 2.14: Sibelius Software [15]

Just like *Finale*, it supports keyboard and mouse input as well as *MIDI* input, but in this case, *Sibelius* provides a small virtual keyboard built in the application. This software allows the user to input their voice to a composition and allows reproduction to check what the user is composing whenever they want.

Another interesting aspect of this software is that it allows the composers to sell the music they created with the software directly from the application.

Apart from the standard software (Figure 2.14), which has a price of 70€ per year, the company offers a simpler version of the software completely free. They also have the Ultimate version of the software for a price of 149€ per year, which includes many extra functionalities aimed at professionals.

Sibelius is also a very professional and complete software. Nevertheless it also lacks accessibility for visually impaired users, which most of the more competitive music notation applications do.

2.3.3 LilyPond

LilyPond is a *free software* application for editing scores [16] that aims for high-quality score production. It is part of the *GNU* Project and it produces traditional ink scores via their own compiling system.

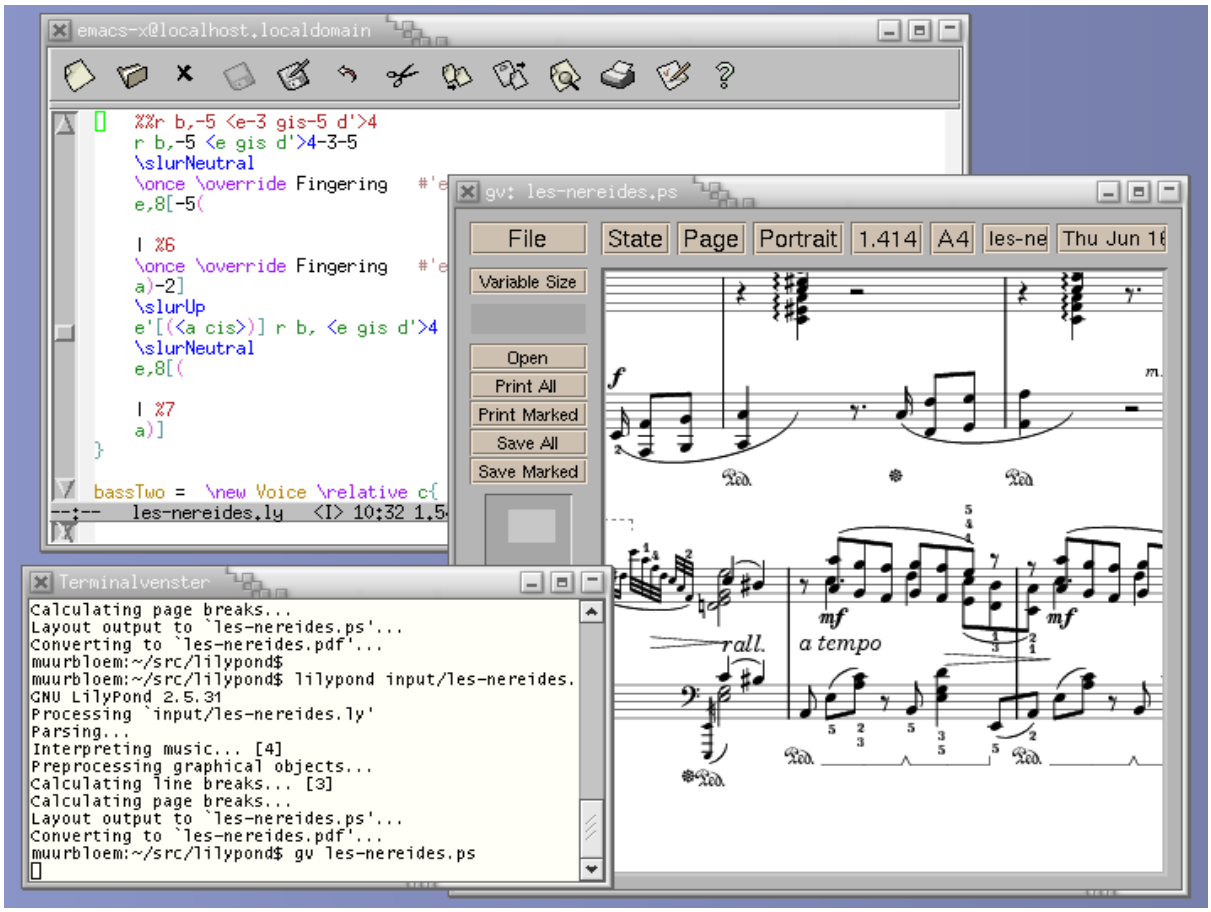


Figure 2.15: LilyPond Software [16]

The program is available for *Windows* and *Linux*, it is one of the few score editors available for *Linux*. Being part of the *GNU* Project makes it completely free to use for everyone.

This software is very unique because while the previous software allows the users to compose music using UI interfaces that let them choose the desired music elements,

LilyPond takes a text file (Figure 2.15) with a specific format and compiles it. The result is then presented on the screen or can be printed directly.

LilyPond is, in its own way, almost like a programming language more than a music score production environment.

Despite having an input method based on keyboard, which is more accessible to blind people than using the mouse on screen UI elements, *LilyPond* can not be considered an accessible application for people who are visually impaired. *LilyPond* outputs musical scores only in ink format, therefore it is not suitable for blind people to check the results of the music they want to create.

The most competitive music notation editors are not accessible, and the accessible software that exists are either very limited or only suitable for people with complete control over the Braille language. For these reasons continuing to develop *LiveDots* is important, in order to provide blind users with an application that is both competitive and that allows people who are learning Braille to also be able to use the software.

Chapter 3

Work Methodology

Our communication has been horizontal, and all the decisions were taken by consensus.

3.1 Project Management Methodology

The methodology we leaned towards is *Agile*, which is a project management methodology characterized by building products using short cycles of work that allow for rapid production and constant revision [51].

Since our project was based on improving and expanding an already existing application. We followed some of *Agile*'s main principles, which are the following: "Our top priority was to be able to deliver something show-able to the customer as early as possible, receive feedback and keep a continuous delivery of software; setting the milestones to a shorter period, preferably a few weeks maximum, the main goal is to deliver working software frequently [52].

We organized our meetings every two weeks, sometimes we would make a meeting in the week in between if we needed to, every meeting or call consists of one cycle. In those meetings, we would show our progress, discuss the roadblocks each group member encountered and identify what were the possible solutions.

Agile methodology is particularly good here since it allows us to get constant revision and feedback on the product. These meetings were usually made with the revision of our professors (customers), where they would check our progress, give out their opinion and ask for changes or additions to the project. Due to COVID-19, almost every meeting was made online.

3.2 Trello

We used *Trello* [53] to keep track of the tasks and the times we had for each, as shown in the Figure 3.1. We followed a simple three-tab structure. The First one was for to-do tasks but was not started yet, the second tab was the tasks in progress, and the third one is where all the completed tasks are located. Our main task was pretty different from each other, and we did not need to rely on work from anyone in order to advance

with our part. So in this case, we did not have to keep a very strict version control or monitor anyone's progress, nonetheless, we still checked that everything was working from time to time and that combining everything would not cause any problems. Version management

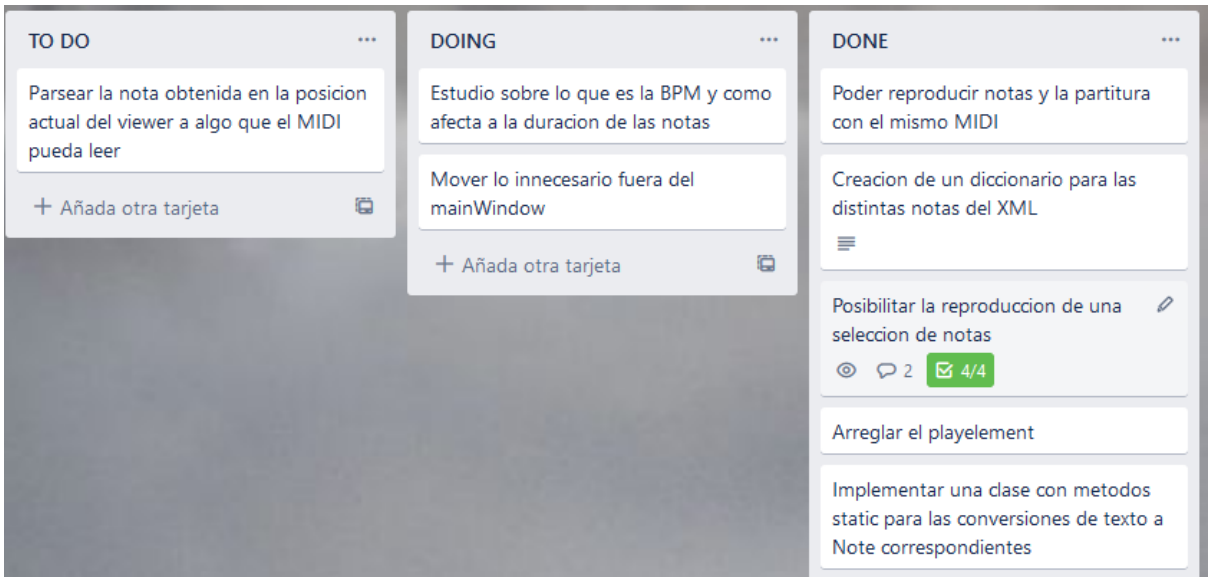


Figure 3.1: Our *Trello* work space.

For version management control, we used *Git*. This would allow us to work simultaneously from a certain iteration by using different branches so there would not be any issues with any progress made since they are independent of each other.

Once the mark is hit on a branch, and only if there are no issues in the branch itself, it can be included in the main branch by merging it.

3.3 Used technologies

We used *WPF*, *C#* and *Manufaktura*

3.3.1 WPF

Windows Presentation Foundation (WPF) is a UI framework that allows the creation of desktop client applications [54]. Being part of *.NET* makes the application's compatibility better among different systems.

WPF separates the visual part of the programs from the functional parts, making it easier to design the UI, in this case, to design an accessible UI for our application.

For the visual part, *WPF* uses *XAML*, a descriptive programming language used specifically to write interfaces mainly for Windows but also mobile applications. *XAML* is also used in frameworks like *UWP* and *Xamarin* Forms.

For the functional part of the program, it uses C#

3.3.2 C#

C# is an object-oriented programming language that is part of *Microsoft's .NET* platform [55]. One of the most important aspects of this language is that it supports multiple types of polymorphism, exception handling, and interface implementation.

C#, being one of the main pillars of *.NET*, is the biggest reason it was chosen for the creation of this project since we need to work with *WPF* Framework which uses *C#* for the implementation of the functionality behind the UI elements.

This programming language is also very convenient for the modularity aspect of the project. Since this project was already started when we commenced working on it, and our work is also going to be part of an even bigger project, everything needs to be divided into different modules so that each part of the program can be modified and improved with ease.

3.3.3 Manufaktura Controls

Manufaktura Controls is a group of *.NET* libraries, written in *C#* used for rendering music notation on different platforms, *MusicXML* parsing, and *MIDI* playback [56]. It supports platforms like *WinForms*, *WPF*, *Silverlight*, *Silverlight for Windows Phone*, *ASP.NET MVC*, and *Windows Runtime Universal Apps*.

In our project, it is used to import *MusicXML* scores and render them to see them as they would be seen on paper, at the same time it allows for limited real-time modification of the notes on the music staff and gives us the tools we need to reproduce the music using *MIDI*.

Chapter 4

Development of the extensions in the application LiveDots

4.1 LiveDots Application

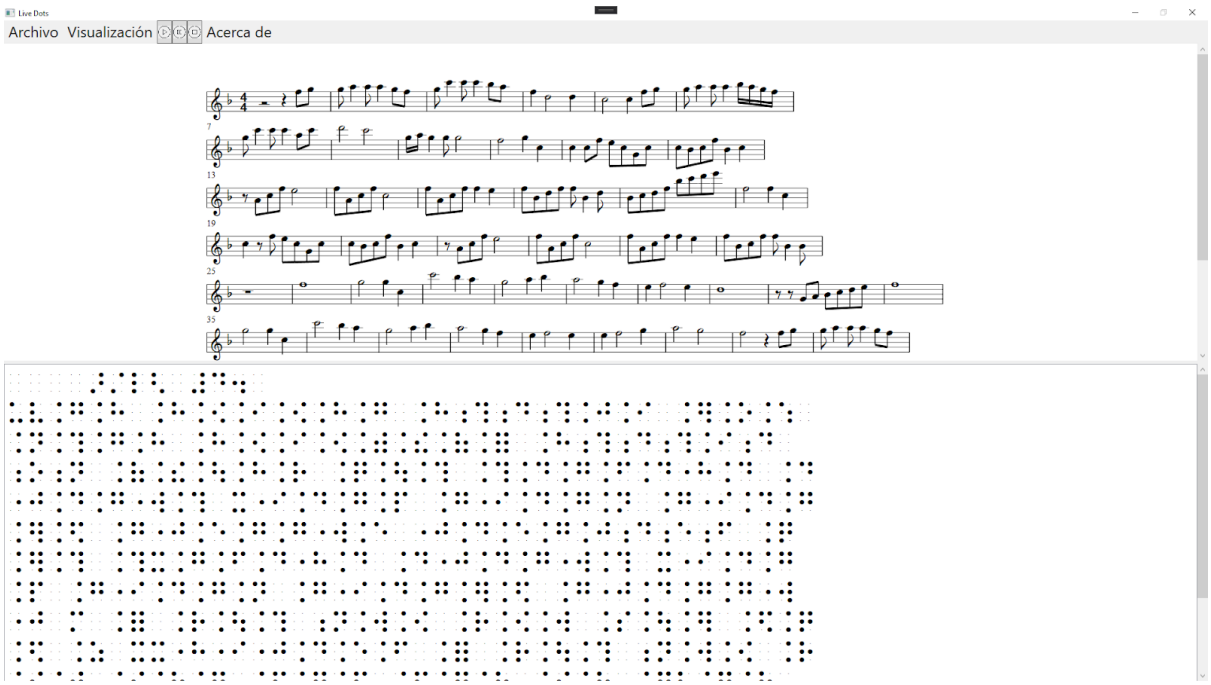


Figure 4.1: General layout of *LiveDots*

LiveDots (Figure 4.21 is an accessible musical application with multiple functionalities such as braille representation of *MusicXML* and *MusicXML_Braille* files, translation from *MusicXML* files to *MusicXML_Braille*, and vice versa, a complete reproduction of the musical score, reproduction of a selected part or note by note, and complete modification of the braille score in real-time.

4.1.1 Application design and functionalities

The *LiveDots* application design is divided into three main parts, which are the menus, the ink score, and the braille score. Each of these three main parts of the application gives the user access to all the different functionalities in the application.

Menu Panel

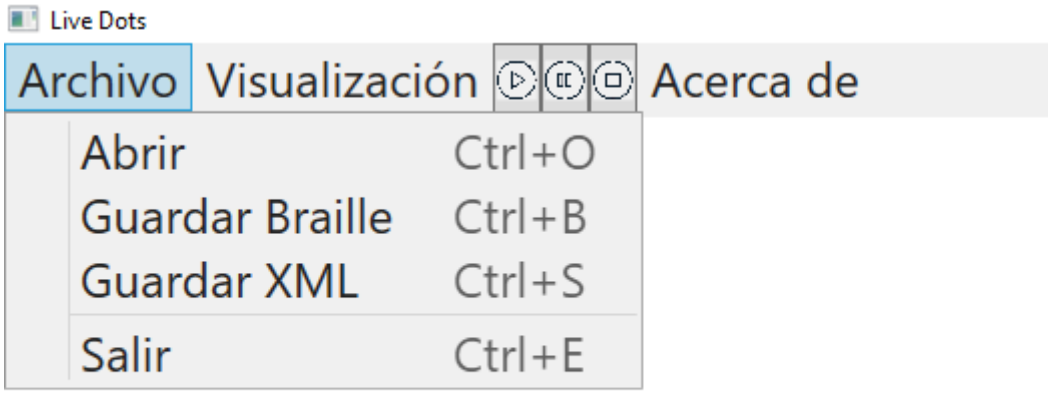


Figure 4.2: Menu screenshot

This part of the application is where the user can access all the file managing (Figure 4.2) functionalities of the application, such as loading *MusicXML* or *MusicXml_Braille* files and saving the musical score to both of these file types.

From the menu, the size of all the panels can be adjusted so that partially visually impaired users can set everything as they desire in order to be more comfortable with the application.

There is also a simple player that allows the user to listen to the musical score from start to finish.

Ink Score Panel

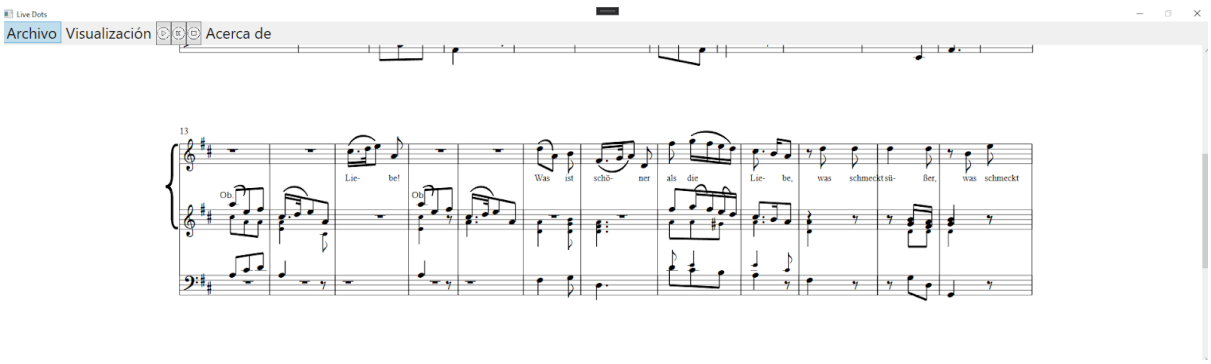


Figure 4.3: TELEMANN.musicxml on ink score

The ink score panel (Figure 4.3) is designed for users who are visually impaired but still have the ability to read directly on the screen. In this panel, the musical score is represented with the conventional ink score method and the score can be modified by making use of the mouse.

Braille Score Panel

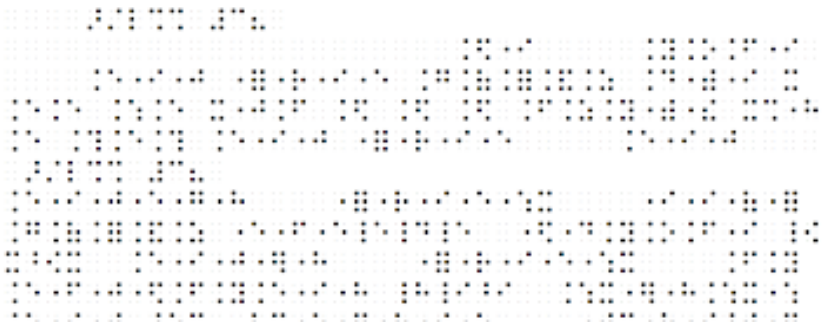


Figure 4.4: TELEMANN.musicxml on braille score

The braille score panel (Figure 4.4) is one of the most important parts of the application since it is both useful for visually impaired users who know braille and prefer to read with braille dots instead of using the ink score and, to facilitate the interaction between a blind user and a sighted user who is using the application together (Ex: teacher and student).

The braille score is both outputted in this panel and to the braille line, therefore while a blind user is reading on the braille line, a sighted user can check what the blind user is doing at all moments.

This panel also has a reproduction functionality, the users can reproduce notes one by one by navigating on the braille score. They can also reproduce a group of notes by selecting which notes they want to listen to with the cursor.

One of the main functionalities of this panel and one of the main features we implemented for this project is to be able to edit the braille score in real-time. People who know braille can modify, add or remove any notes they want to create their own music.

4.1.2 Use cases

To explain the use cases of the *LiveDots* application we are going to divide the different types of users into three types which are:

- **Blind users**: people who are completely blind.
- **Partially blind users**: people who are visually impaired but not completely blind.
- **Sighted users**: people who are neither blind nor partially blind.

The following graphs represent the use cases of the application for all its functionalities. We decided to divide them into four big groups: file managing, screen view, music player, and braille score.

File Managing

File managing involves all the functionalities which deal with loading different types of files and saving the musical score to different types of files.

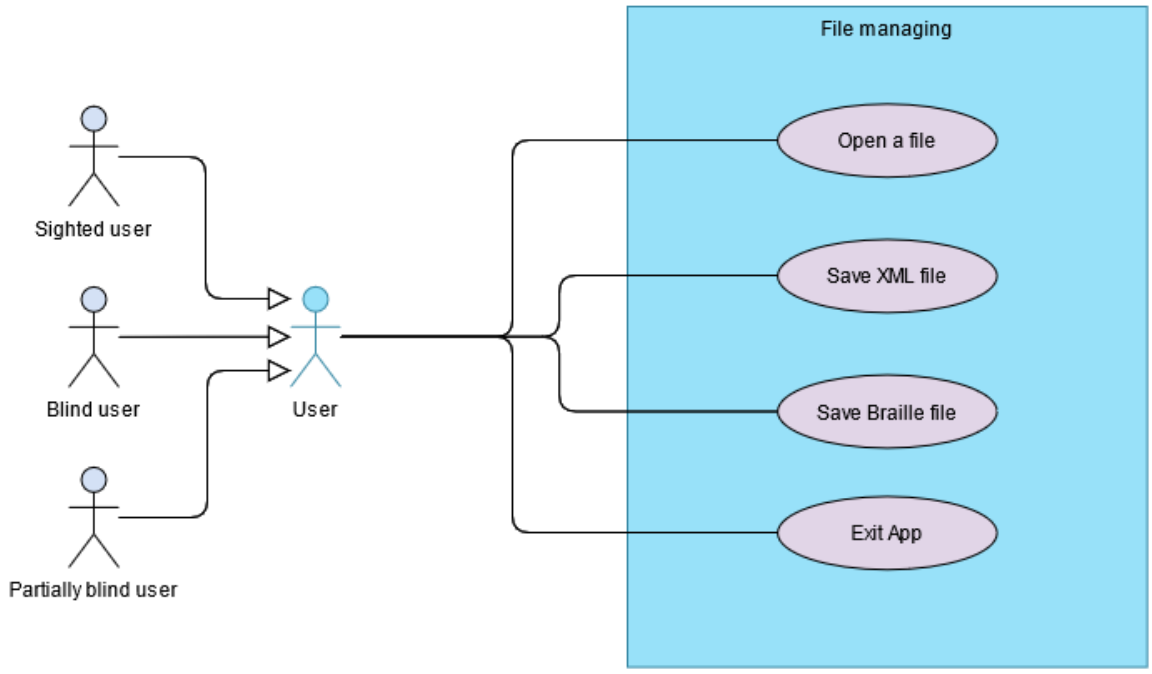


Figure 4.5: File managing use cases diagram.

All the functionalities involving file management are suitable for all three types of users so, in this case, the user type named "User" (Figure 4.5) is a combination of the other three types of users.

The user "User" will make use of:

- Open a file: opens a file from the user's computer. This file can be in *MusicXml* or *MusicXML_Braille* format. The file is then converted into a musical score.
- Save *XML* file: saves the musical score into *MusicXML* format.
- Save *Braille* file: saves the musical score into *MusicXML_Braille* format.
- Exit app: closes the application.

Screen View

Screen view deals with all the functionalities concerning adjusting the size of all the UI elements of the application.

In this case, users who are completely blind will not make use of this functionality, therefore the user named "Non-blind user" (Figure 4.6) is a combination of the users who are visually impaired but not completely blind and the users who are not visually impaired.

The user "Non-blind user" will make use of:

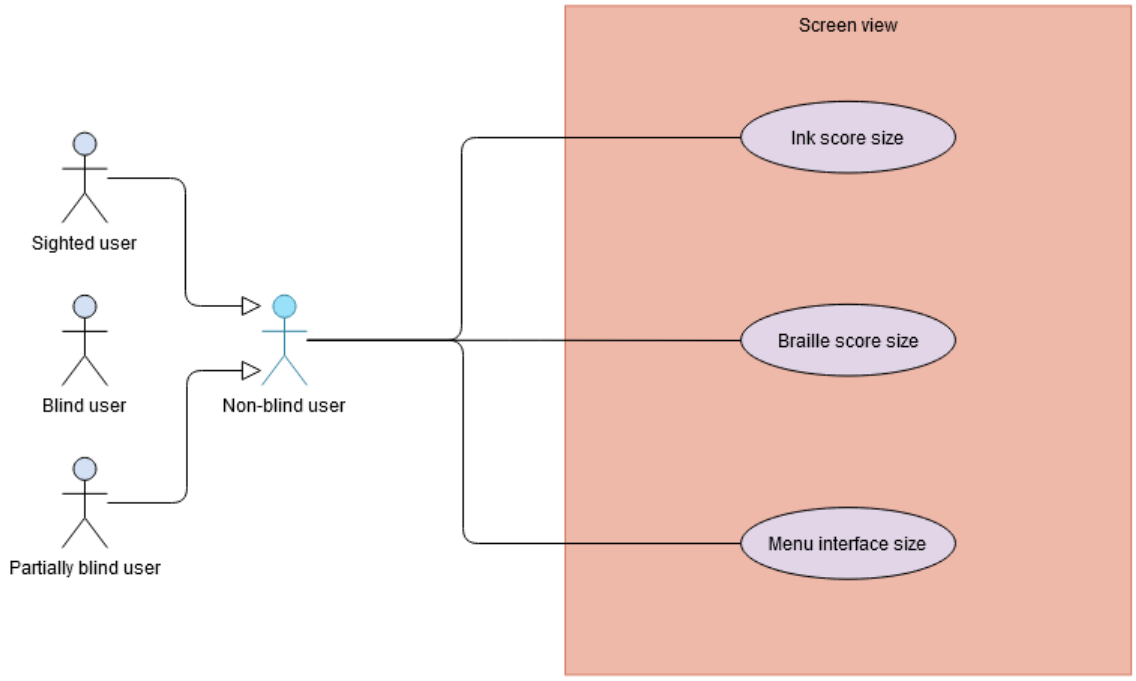


Figure 4.6: Screen view use cases diagram.

- Ink score size: increases or decreases the size of the elements in the ink score.
- Braille score size: increases or decreases the size of the elements in the braille score.
- Menu interface size: increases or decreases the size of the font in the menus.

Braille Score

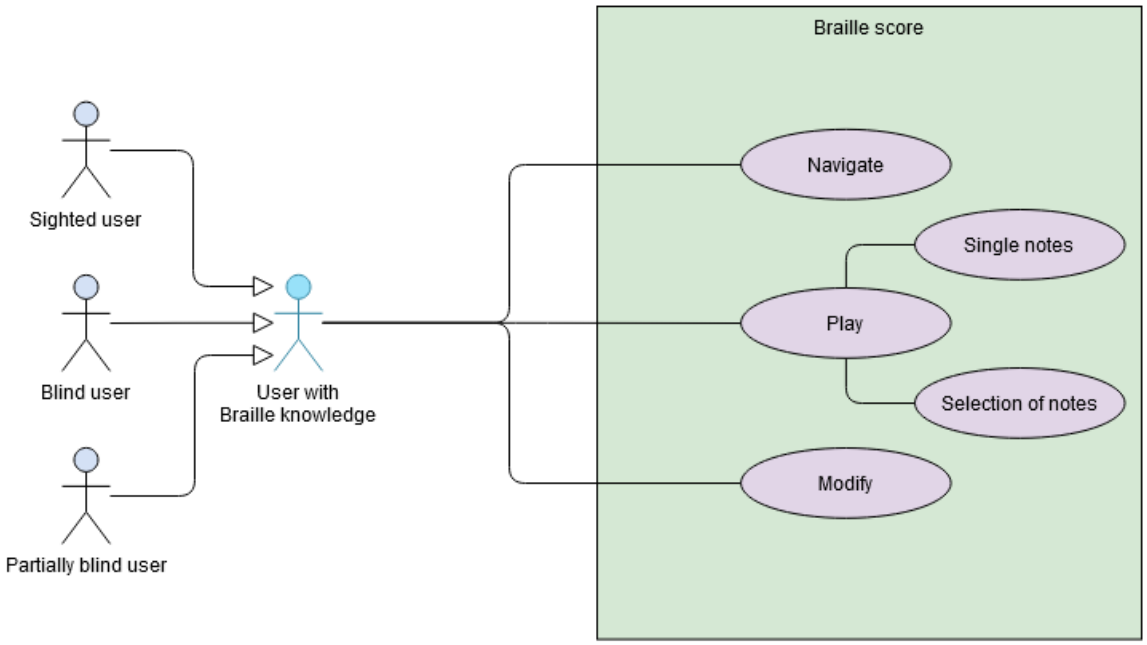


Figure 4.7: Braille score use cases diagram.

We called Braille score to the group of functionalities that are accessible from the braille score panel.

To make use of these functionalities (Figure 4.7) the user needs to know braille, therefore for this case we created a type of user called "User with braille knowledge" which involves every braille connoisseur from any of the three types of users.

The user "User with braille knowledge" will make use of:

- Navigate: the user can use the arrow keys to navigate in the braille score, making use of JAWS the user will get feedback of the musical elements they currently have the cursor on.
- Play: when navigating the braille score, apart from the screen reader feedback, in case the user does not use one, the application gives feedback by reproducing the sound of the note the user has the cursor on. The user can also play a group of notes directly by selecting them together.
- Modify: the user can modify the braille score as they wish, the changes they make will apply also to the ink score.

The braille score is the most important of the two scores because the ink score is mostly used by users who are not visually impaired in case they want to check and compare it with the braille score.

Music Player

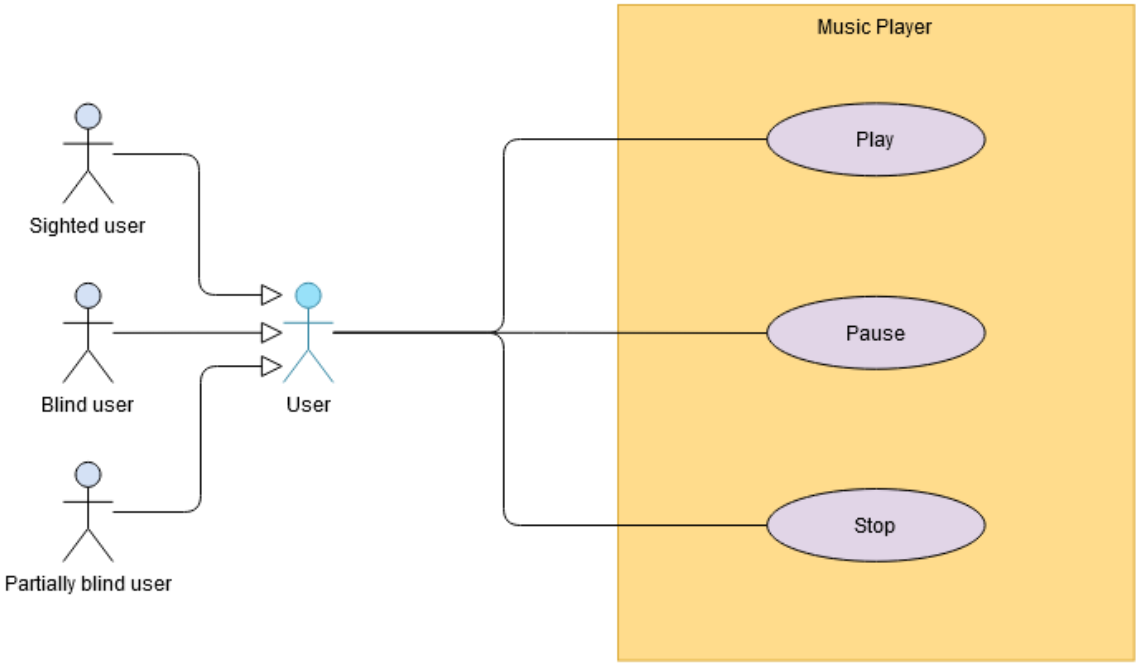


Figure 4.8: Music player use cases diagram.

Music player involves all the functionalities from the music player which every type of user makes use of. The users can, as seen in 4.8, play the musical score, pause and stop it.

4.1.3 Software Architecture

Before diving into the detailed explanation of the parts we implemented, we can see the fundamental software structures of the application below in Figure 4.9 and Figure 4.10.

The first picture shows how the main structure is, along with the more relevant functions or variables it contains, the application has a *MainWindow*, where some of the important events that occur in it are handled by *LiveDotsCommands*, important events are eveything related to the interactions that the user has with the app, such as the pause button or the increase size button, other events such as cursor movement is handled by the class itself, however, the extra functionality is handled by *CursorPosSound*, this class has extra resources that it uses for the different functions implemented in the application, such as a Dictionary that provides instant translation from Braille to *Note*.

The Braille text viewer is handled by *BrailleMusicViewer*, it uses multiple classes to build up the final result, which is showed in the second half of the screen, and lastly, *MyMidiTaskScorePlayer* controls all types of musical reproduction, handling the sound, including the durations, pitches of each musical element...

In case any XML file needs to be parsed, *MusicXMLParser* handles it. *Domain* has everything that *MusicXMLParser* needs to store all the different data structures *MusicXML* uses, further details are shown in Figure 4.9.

The second picture shows the tree structure that will be explained further below how it functions. This tree structure is used to convert a Braille musical score back to *MusicXML* format, this structure is the same when the application converts *MusicXML* into Braille music format, the classes that are used are all related to the different elements contained in a music score.

Although Figure 4.10 only shows the structure of the classes needed to convert from braille to *MusicXML* the structure needed for *MusicXML* to braille is similar to the changes in the logic of each class. Both are connected to the class *LiveDotsCommands* by the use of factories.

A much more detailed class graph can be found in **Part C Appendix: Detailed class diagram**, which shows the main program and both of the trees, including the functions and the variables that are used in each.

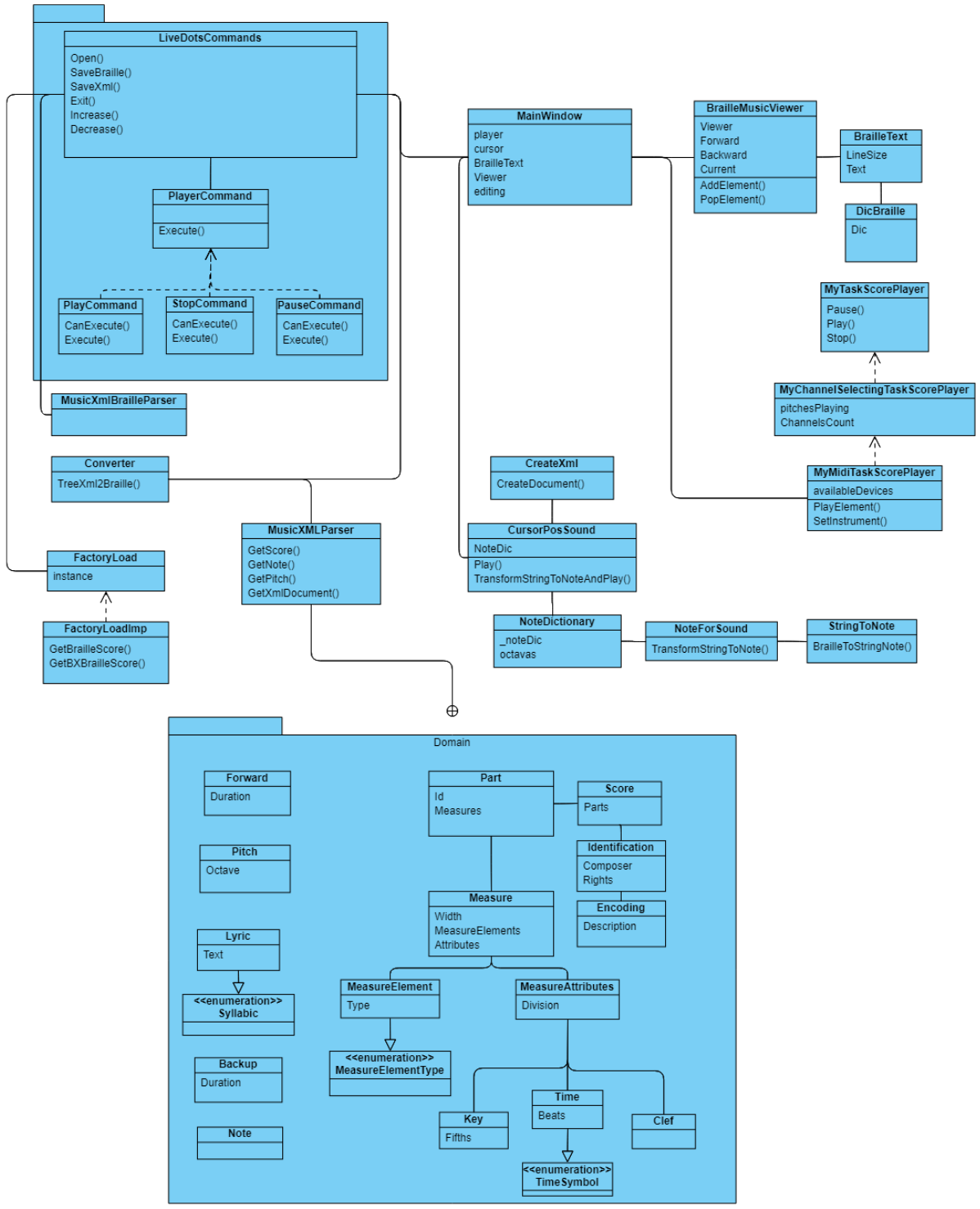


Figure 4.9: Class diagram of the application *LiveDots*.

4.1.4 Our job improving LiveDots

The main objective of the project is to expand the functionalities of the application and the interaction for the sightless user, so the focus was on the braille musical score. The application was already able to graphically represent the content of the braille musical score in the display of the user, moreover, the user was allowed to move freely in it.

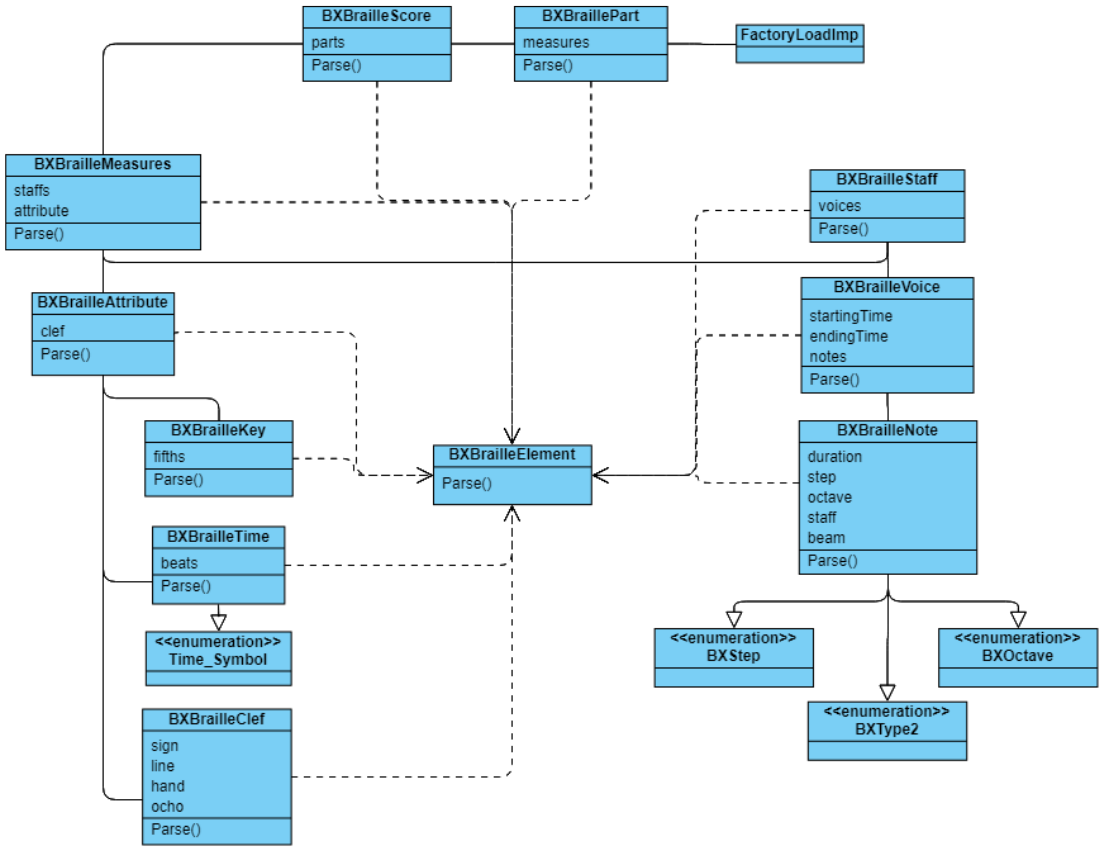


Figure 4.10: Class diagram of the application *LiveDots*, specifically the Braille to XML.

However, the user would not understand the musical notes if the user did not have previous knowledge whatsoever about music; adding sounds in the position of the cursor would help the user with no previous knowledge, allowing them to hear what musical note is and match the sound with the corresponding braille musical note. But, what about the more experienced user, the answer was giving the sightless user the option to edit the braille musical score live, granting them the possibility of adding their touch to the musical score. This would also open up the possibility of starting a musical score from zero and making a whole song within the application. All of this will be explained further below this, along with its development process in detail.

4.2 Translation from Braille to MusicXML

One of the objectives proposed to upgrade the application was the implementation of functionality in which an xmlBraille file could be selected to be shown on the screen and saved as an *XML*. The code was already capable of opening an *XML* file parse it to a braille format and visualizing the ink score and braille form.
Since the new part was going to do the same but in reverse, we opted to keep a similar structure of what was already implemented but with a few changes, the structure follow

can be seen in the figure 4.11. To do this process we first started by dividing the logic when a file was selected so depending on its format a correct form of parse was done. In our case, we read the *XMLMusic_Braille* file, saving it in a list which we will use in the parsing process. This process fills a tree structure and obtains the score in its ink and braille form needed to show both scores by *Manufaktura*.
With the tree fully completed the next step is to create a string containing the score in an *XML* format so the user could save the ink format score.

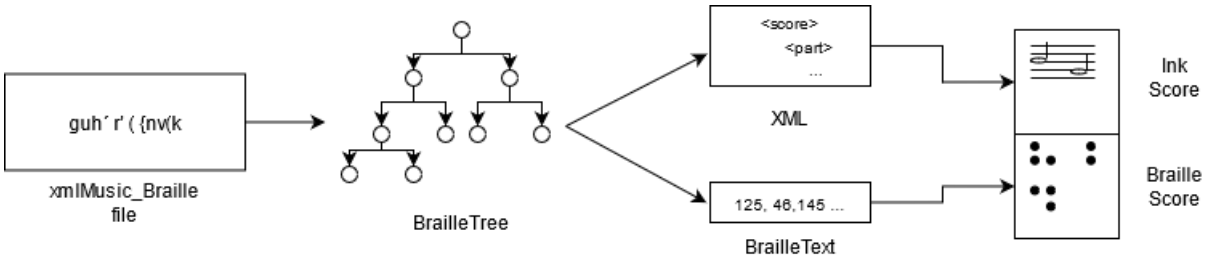


Figure 4.11: Scheme of the parse.

4.2.1 Braille file

The file contains the braille score coded in *ASCII* characters, each character has a braille sign linked to it.
The initial idea was just to go reading char by char and parsing it to its corresponding value but during the implementation, we run into the problem that some values like the alteration of a note required an irregular number of characters , meaning that every parse of the notes of the score will use a different number of characters from the file thus, this method was unpractical due to its unpredictability.
To solve this, we opted to use a list of char, and every time we parsed a character it would be removed from the list.

4.2.2 Braille tree

To create the braille tree we decided to follow a structure resembling how a score is structured. The structure can be seen in the image 4.12.
The braille tree is composed of a primary class named *BXBrailleScore*. This class will represent the score as it can be seen in the figure. A score is divided in parts, each part is a score for only one instrument. To show this representation in our tree the class *BXBrailleScore* contains a list of *BXBraillePart*.
Each part is divided in measures, a measure containing a determined number of notes. We divided the measures into two components, the attributes of the score and the staff. To represent our measures we use the class *BXBrailleMeasure*. This class contains a reference to BXBAttribute and a list of *BXBrailleStaff*.
The class BXBAttribute is responsible for the parse of the attributes of the *partiture*. The attributes are divided in three:

- The clef indicates the position of each note in the staff. For example, as can be seen in the figure 4.13 the note G in the staff with Treble Clef is in the second line meanwhile, in the one with AltoClef the note is between the first and second line.

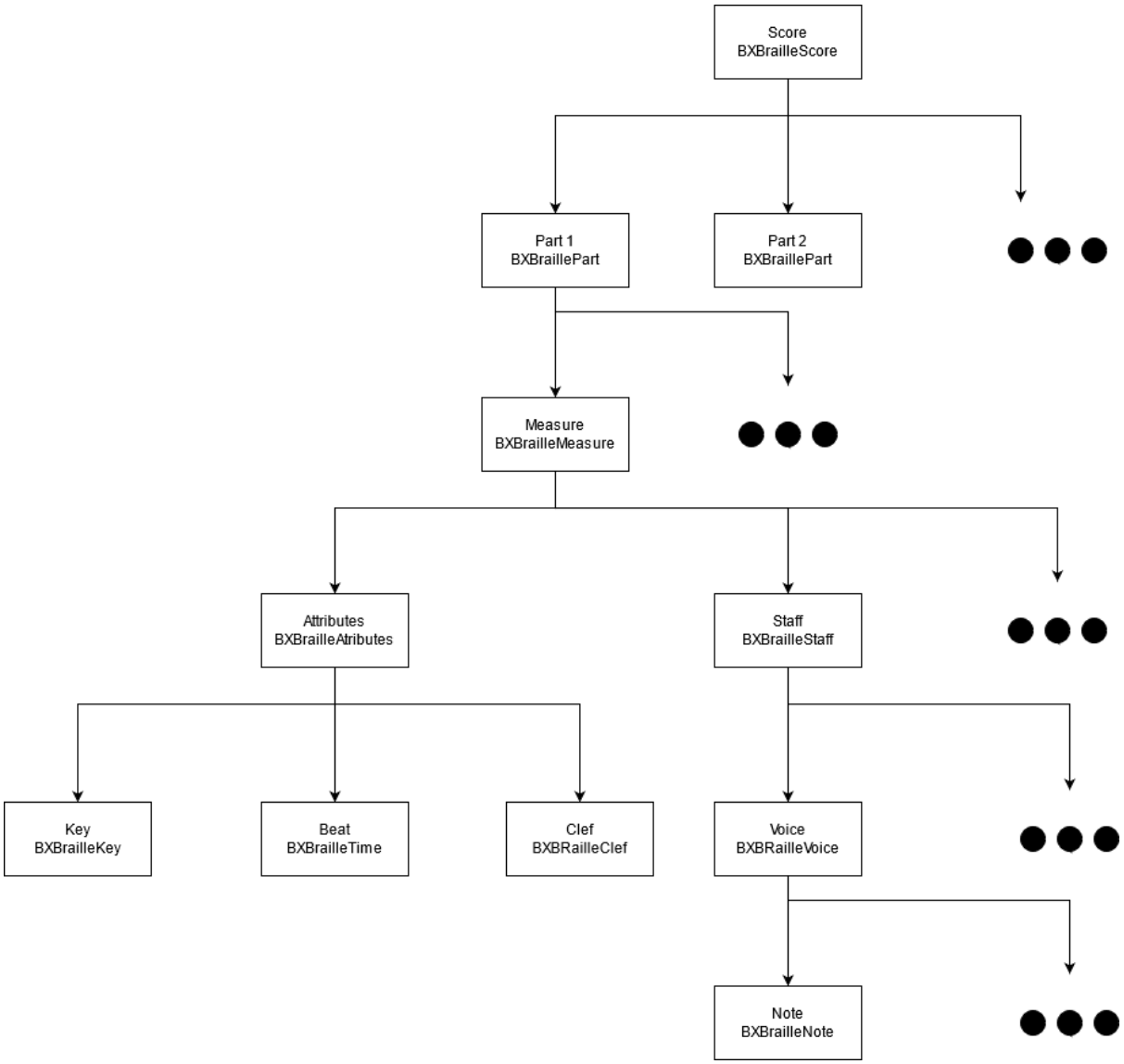


Figure 4.12: Structure of the braille tree.

This is implemented in the class *BXBrailleClef*. To do it we read the first character of the list *BrailleList* to obtain its corresponding sign and we do the same with the next character to obtain the line. The third character is used to obtain the hand, right or left, and the last two characters are used to know if the octaves are higher or lower than usual.

- The key of the score is the group of pitches. We use *BXBrailleKey* to decode the fifths and mode.
- The beat indicates the number of notes per measure, since every type of note has a value. This is done in the class *BXBrailleTime*, it uses the "BrailleList" to decode the compass and the beat.

Once the attribute had been decoded, we passed to the second part of the measures, the staffs The staff is the five horizontal lines in which the notes are drawn. This is done in *BXBrailleStaff* that contains a list of *BXBraillevoice*.
The voice is a single strand or melody in a composition. In the code we use the class

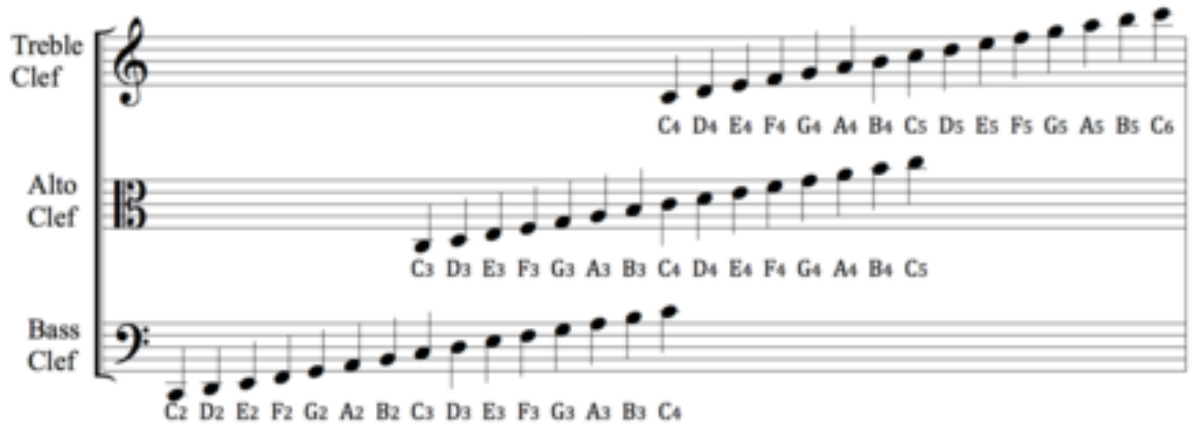


Figure 4.13: Position of notes depending on the clef [17]

BXBrailleVoice to represent it.
BXBraillevoice have a list of *BXBrailleNote* The notes of the score are decoded in the class *BXBrailleNote*.
BXBrailleNote is the class that contains all the information of each note. The first step is to see if the note is a rest. In that case we use the character to parse one of the four types of rest and we fill the class object with the type and a Boolean variable indicating that is a rest for future use. If it is instead a note we first see if it has alteration by using two or one characters in the list then comes the octave in this case if no character match with one of the cases we use the default octave that been the fourth. Last is the step and type of the note. To parse these two properties only one character is used additionally if the type is quaver a boolean variable is set true to indicate that the note uses a beam to connect to close notes.

4.2.3 BrailleText

During the filling of the braille tree, we have been adding the braille representation to a list contained in the class *BrailleText*. Since the information stored in the file was just characters we need not only to parse its ink representation but also know the dots that represent the braille form. The braille representation consists of a string form by the position of the dots. The figure 4.14 shows an example of how the notes are represented in brailleform.

4.2.4 Translation from Braille Tree to XML format

To save the ink score to a *MusicXML* format we needed to transform the information in the braille tree to an XML. At first, we tried to serialize it with the library XmlSerializer[57], this library allows one to create an XML following the structure of a given class and the inverse, a class following the structure of an XML. But the resulting XML had a structure that *Manufaktura* could not read so we opted to create a new class in charge of creating the XML.
To fulfill this task we first start by writing the tag relative to the score adding one for each *BXBraillePart* in the program. Encapsulated in his tag is the measured tag with the number corresponding to each *BXBrailleMeasure* position.

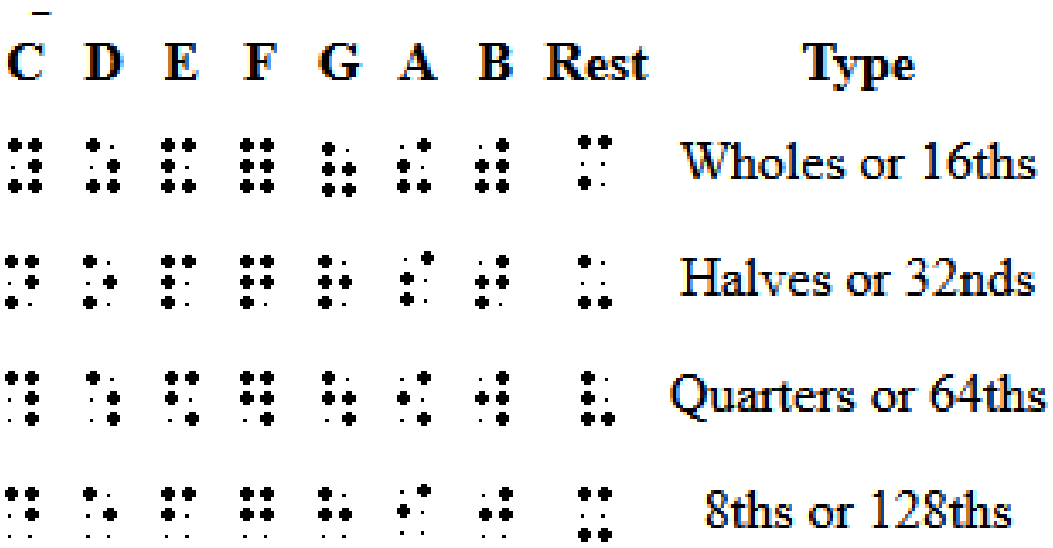


Figure 4.14: Representation of notes in braille.[18]

The tags corresponding to the attributes this is to say the sign, line fifths, beats, and beats type. This tag is only included in the first measure since all others will use the same attributes. In each measure tag, some notes depending on the beats of the score are contained. The notes are added differently depending on if the note is a rest. This is done using the boolean variable mentioned in the "braille tree".
For the rest, a tag is added with the duration, voice, and type of rest. For the notes, the pitch is added with the step and the octave, next is the duration, the voice, and the type. The type of steam is selected following the octave of the note and the tag "beam" if the note is connected by its steam to others.

4.2.5 Visualization

With the brailleText and the XML file completed now, it is time to show the ink and braille score in the program. For the ink score, the open code library *Manufaktura* is used. This library uses an *XmlMusic* file to create a visual representation that will be shown in the top part of the view. For the braille score, we will use the list containing the position of the points of the attributes and the notes. p

4.3 Reproduction by cursor position of the braille musical score

This part aims to reproduce the corresponding sound of where the cursor is located in the braille music score. The application's user interface is divided into two halves essentially, on the first half of the screen, an ink musical score is displayed and the other half is the braille musical score, the cursor movement is located in the second half, so the sightless user can feel what musical note wherever the cursor is at. We will cover every element used to accomplish this function.

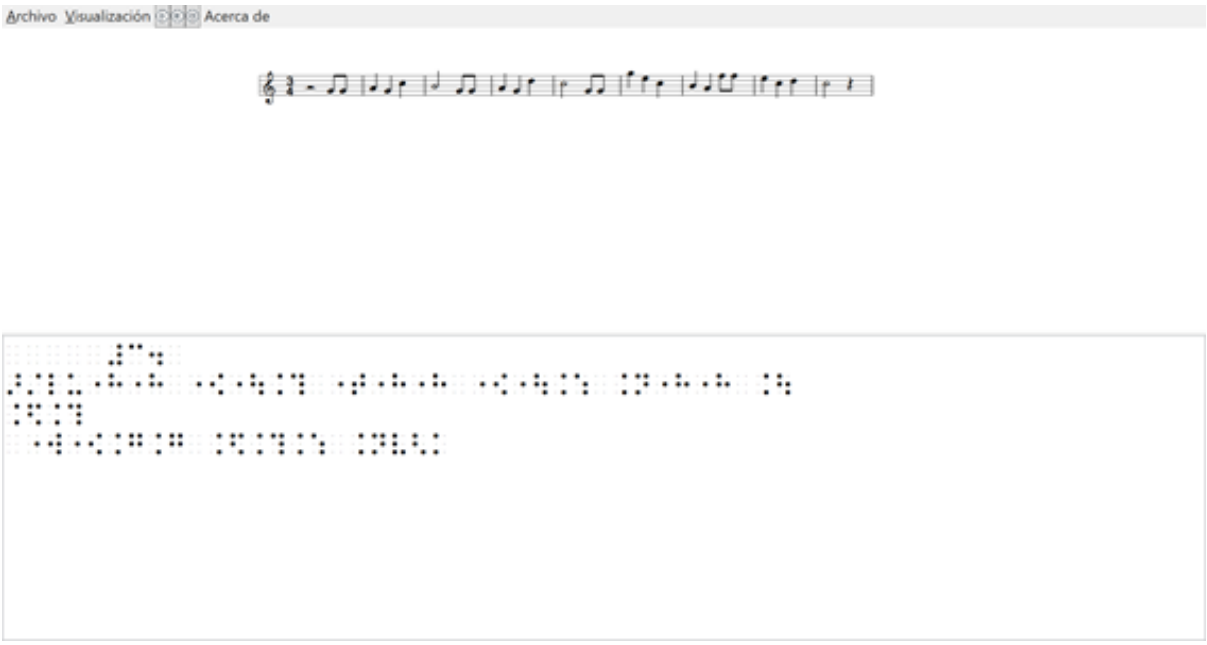


Figure 4.15: Visualization of the scores

4.3.1 Movement around the braille musical score

Cursor movement around the braille text is accomplished using the XAML technology, the braille musical score is essentially a read-only (at first, this will be modified later, as we are increasing the functionality of it) *TextBox*, which means that any text inside this box can be selected and also the user can move around the text box freely. Internally, the braille is a combination of many symbols that are not legible, these symbols have meaning in braille and they are displayed as dots of different combinations for the blind user (Figure 4.16).

A translation from those symbols to a readable format is needed, which is something that was already thought about and implemented. The *Textbox* is located in the second half of the display and it contains the braille musical score, the user can position their cursor anywhere in the text and run through all of them with it; internally, the *TextBox* has each "box" of braille mapped to an array where the musical note is written in Saxon musical notation format, this "box" is also defined by the symbols that are not readable which we mentioned before. Each of the symbols corresponds to a certain combination of dots in Braille.

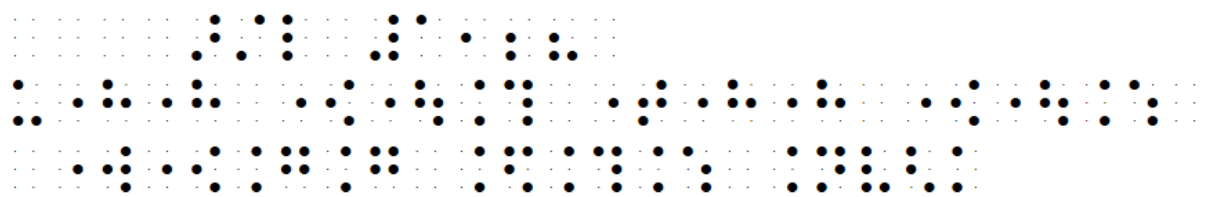


Figure 4.16: Portion braille musical score of *Happy Birthday*.

The class that handles all of this is called BrailleMusicViewer, which compares the real

position of the cursor in the application and the position the class has saved locally. This class contains two arrays that handle the size of each Braille "box", one array contains the distance to the next note on the right or in the next line and the other one contains the distance to the left or the previous line [2]. With those data structures, the application can find out if the user has gone forward or backward in the *TextBox*.

After moving the cursor, a comparison is done between the locally saved position and the new one, and it is contrasted to see if the new position is on a different note (remember that in braille format, one box does not necessarily equal one musical note) [2]. And lastly, the local variable of the cursor position is changed and updated. What this means is that if we want to recover the cursor's position, it will return the musical note it is pointing as a string, which is what we will use to convert into a musical note format that *Manukaftura* can understand and produce a sound of.

Understanding how to use the Manufaktura library

Now, we need to figure out how to convert the string with the musical note written in text literally into something that can be reproduced. Our first thought was to create a new music sheet and insert the different musical notes into it whenever we wanted to reproduce something. This seemed the option but a music sheet needs many more elements, like a key signature or the amount of staff, which are things that we have almost no knowledge of. In addition, Manufaktura uses different channels to reproduce music sheets, which meant that we might need to tweak their way of reproducing if we wanted to introduce musical sheets into the equation. It could over-complicate things so we decided to look for another option if it was possible, also it would not make much sense to create a music sheet for a single note.

After some investigating, we discovered that the Manufaktura library supports single note reproductions (one element), meaning that you could send one individual musical note and it would only reproduce that note with its corresponding rhythmic duration, which also needed to be defined beforehand. This is exactly what we were looking for, so we went ahead and decided to pick this route.

4.3.2 The transformation from a string to a playable item

As said before, the string was returned, but it was far from being something usable. First, we would need to simplify it and make it more consistent so that the different musical notes would be shortened and treated equally. The way to achieve this is to use a different type of musical notation. The currently used system was the Latin convention (used mostly in Latin-speaking countries such as Italy or Spain) [58], and this type of notation had an inconvenience for our use: different types of musical notes had different lengths written in a string; besides, different note durations also had different lengths in a string.

To simplify this we chose to change the used system and use Saxon (Anglo-Saxon convention) musical notation convention instead, which used the alphabet to represent the different types of musical notes and we decided to use numbers to denote their durations, for example, A1 means the note duration is one quarter, this will be stored in an XML file format .resx which we will use to retrieve all the different combinations.

Creation of a RESX file

"*A resource file is an XML file that contains the strings that you want to translate into different languages or paths to images.*" [59]. Each pair is an individual resource and their extension is *.resx*. This type of file was what we needed to transform from Latin musical notation to Saxon musical notation.

Another option was to create a local script that translated it or a dictionary to store it, this last one is what a resx essentially is, but with the difference that it much more accessible because it was global and also customizable, and although it could increase the memory load, the size of the file was small, so it was not noticeable. In return, the list of musical notes was much more accessible for modifications or tweakings while also being more readable.

Conversion from a string to a playable note

After successfully implementing the *.resx* file. Now we need to transform the string with the Saxon style musical note, which is the musical notation that *Manufaktura* follows, a conversion is needed since it does not support direct reproduction. The transformation cannot be done in the *.resx* file itself because the variable that was needed in order to reproduce was very specific and had to be done manually. The first version of the conversion was straightforward, this implementation was not too complicated after understanding a bit more the *Manufaktura* library, the conversion was trivial in the end since it already worked with Saxon musical notation. But another problem appeared, we noticed that musical notes in a piano have many different octaves (Figure 4.17), which sound different from each other, these vary between high tones and low tones (each octave has its tone), which are called pitches.

At the same time, one octave has a total of eight notes, a regular classic piano has seven octaves in total, so that sums up to a total of fifty-six different sounds, some more similar to others. To keep the application more efficient, we would need another structure to save all of these different musical notes. In the meantime, we decided to use the fourth octave as a placeholder.

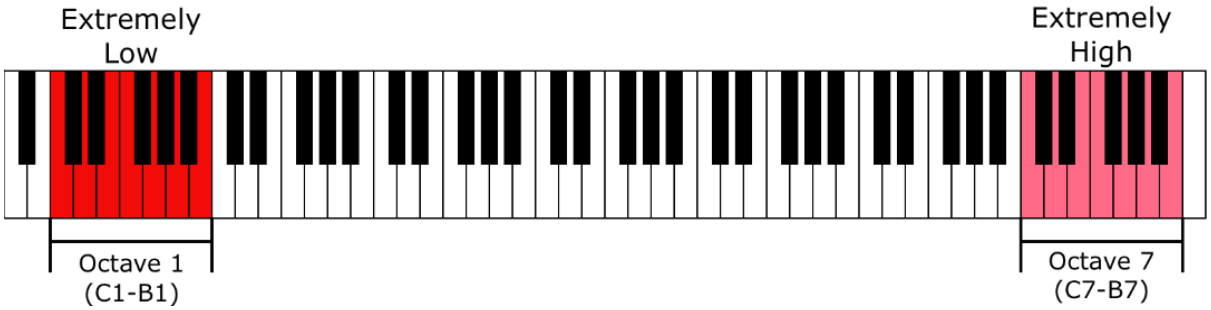


Figure 4.17: The different octaves in a piano [19].

Dictionary to store the different octaves

So, until now, the octave part of the note was not being processed, adding it would mean we would get many more different types of notes to process and transform into a playable element. To check all the different notes, we need a structure that saves up

the pitch directly when given a specific musical note. We decided to create a custom class called *NoteForSound* that would essentially have a variable that was compatible with *Manufaktura*, and an internal conversion from string to that variable. This class would be part of a dictionary, which needs a string and returns a musical note. This way, whenever we needed a musical note, we asked the dictionary with the key (musical note in string format) and the dictionary would return the corresponding musical note if it exists, which is ready to be used for reproduction.

For the creation of the dictionary, besides using the custom class to store the desired elements, we also chose to make an XML file, this was done so that it would be much easier and accessible to edit and expand the range of octaves if it was needed, while also making it more legible for the user, while also making the debugging part easier, since it is something that is handled externally. As of now, the dictionary supports up to seven octaves, which corresponds to the standard configuration of a regular piano. A modern piano can have more than seven octaves.

Full support for the different octaves and note durations

After a successful implementation of the dictionary, this would mean that the first objective is almost completed, the application needs to handle the reproduction properly, so we needed to control it somewhere, the application has a *MainWindow* class, which mainly handles all the different events that happen when the application is running, there is a method that handles the event of the cursor, whenever the cursor moves [2]. So adding our wanted functionality here would mean that our goal was completed.

Previously, we were able to reproduce a musical note with one pitch and one duration. Now, it can reproduce all the musical notes that the braille music score has. All of the note handling part, from string to the actual reproducible musical note, is handled outside of *MainWindow* by the class *CursorPosSound*. This class creates a dictionary and handles the play function. When *MainWindow* detects a new musical note that has not been reproduced (a new cursor position), it will call this *CursorPosSound* and this class will receive the string, convert it and call the play method to make the sound.

4.3.3 Reproducing a selected group of notes

Since singular note reproduction was successfully implemented, we decided to expand the scope of this module by allowing the user to not only reproduce one musical note but a whole selection of musical notes, whatever the user selected with the cursor. The idea would be the following: the visually impaired user can select a group of notes in the braille musical score; after finishing the selection, the application would then play all the musical notes that were selected, following the correct order, from the beginning to the end, and if the user decides to unselect or move the cursor when the previous group of musical notes is still playing, it will stop.

Implementation of the selected group of notes

For the implementation of this, we thought of the same two ideas as before, either creating a musical score from zero and then adding all the selected notes one by one to the musical score; when all the notes are added then we would use it to reproduce what was selected by the sightless user. Doing this would imply transforming the musical staff of the original

musical score and using it, along with many other musical elements that are needed to make a proper musical score. *Manufaktura* needed many variables of music that we did not have knowledge of, so this idea was not liked much.

Another option would be to use the already existing function of playing single musical notes. But this would mean we would need to control each of the rhythmic durations of the different notes to make it sound one by one and not all the notes altogether. This was easily solved since we already had to control and use these durations to play the single musical notes. The idea was to iterate through a list where all the selected musical notes are and then reproduce one by one in order, but we needed to put on hold the iteration through the notes' list when the musical note at the top was being reproduced, and only if it finished reproducing, the next one would come in and be reproduced. Also, we needed a way to notify the application if the user decided to unselect or move the cursor, to stop the reproduction.

After some thorough investigation, a solution was found for the second approach, so the first one was discarded. The solution was to use an asynchronous method, this way when one musical note is being played, the iterator would stop there. This method would allow the user to keep using the application even if the notes were still being played, hence the need to stop the reproduction when the cursor position is displaced in the musical braille score.

To stop the asynchronous function, we used a *CancellationToken*, which is a struct that propagates a notification to cancel certain tasks [60]. This is triggered when the user deselected the previously selected group of notes, either by moving the cursor or clicking somewhere.

4.3.4 Small optimizations in code

After managing to implement the last part. The class *MainWindow* was a bit of a mess, the code was altogether in one function and it was very hard to understand properly. To solve this problem, we separated the different functions into different methods. So now the event handler would only have two lines, one method, *setCancelInSelectionPlaying*, this method checks if the cursor moved and if there was any selection before that, if there was, then stop reproducing the musical notes. And last but not least, the method *ProcessSelectedNotesAndPlay* handled the reproduction of musical notes and the position of the cursor. Here we check the cursor's position, whether it was moved or not. If moved, then reproduce the musical note it is pointing to or if there was a selection done, then call *PlayGroupOfNotes* and proceed to play all the selected notes.

4.4 Real-time modification of the braille musical score

Another of the main goals for this project was to implement a system that would give blind users the ability to edit and modify the musical score efficiently and directly in braille. This would mean a huge improvement in the accessibility and utility of the application.

The application previously had the option for sighted users to edit the ink score by dragging the notes up and down using the mouse (Figure 4.18), which was not intended or

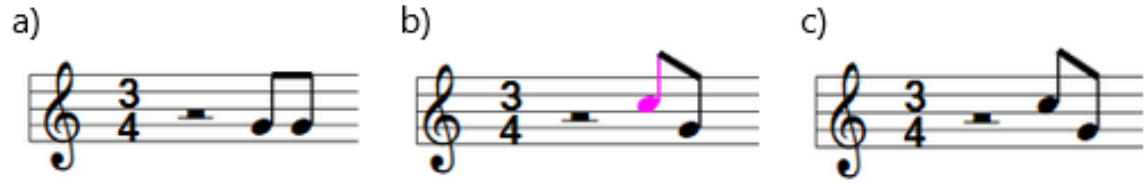


Figure 4.18: Mouse modification of the ink score.

accessible for blind users. This functionality only allowed the sighted users, or complicatedly the blind users to change the tone of the already existing notes on the score, it did not have the option to add new notes or remove existing ones. Therefore, it did not either have the option to add other musical elements such as silences and more advanced elements.

4.4.1 Deciding the best way to implement the functionality

We wanted this functionality to be both accessible and easy to use for blind users who are new to music applications or music in general but also to be useful and competent for experienced musicians and people who are willing to create complex pieces of music using this application. Therefore, we had to think of something straightforward but not limited when modifying the braille score.

Our first idea was to create a system where the application gave the user the option to input new notes by choosing which note they wanted. Selecting them from an interface that would cycle between the notes, reproducing the sound of each of the notes so that the user chose first the tone of the note and then the length of this one.

After thinking about how we could implement this, we noticed that this idea would have a lot of problems since a lot of the musical elements that the application needs to be able to input in the score are either silent notes or elements that do not necessarily note themselves.

Another problem was that this system would be really slow since the user would have to choose each element one by one and go through at least three different steps to input just one element. Moreover, this would probably be hard to use with the braille line, since its purpose is to show the text that appears on the screen, not a complex interface.

Taking into consideration these inconveniences we chose to discard this idea and work in a new one, one that was faster to use and did not have problems with elements that are not regular notes.

Once we were sure that the previous idea was not viable, we started thinking about new ways of implementing this functionality. Learning from the previous mistake we decided that the best way for modifying and editing the music braille score was going to be by directly inputting the desired elements by the keyboard.

4.4.2 Direct keyboard input method

The decision was finally made, now the problem was designing how this method was going to work since there were some drawbacks that we had to solve.

How blind users were going to input the music elements

First, we had to know how blind users usually write music using the braille line, because we doubted two different ways.

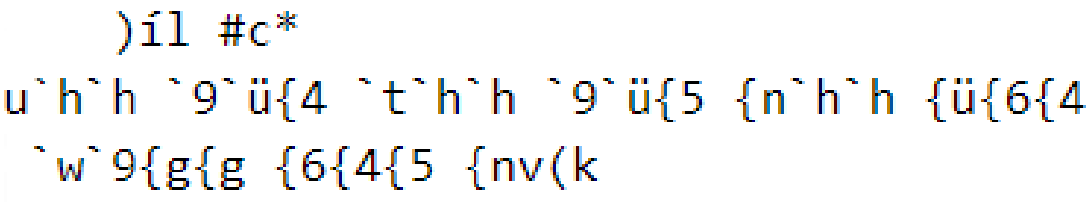


Figure 4.19: Happy Birthday in *MusicXML_braille* format.

The first way was using the braille line to directly input the *ASCII* characters that represent musical elements in braille (Figure 4.19). This is how *MusicXML_braille* files work and it would make sense that blind users input elements like this.

In this case, implementing the functionality was going to be easy, since we already had a parser that translates *MusicXML_braille* to *MusicXML* files, which we could use to implement the editing functionality.

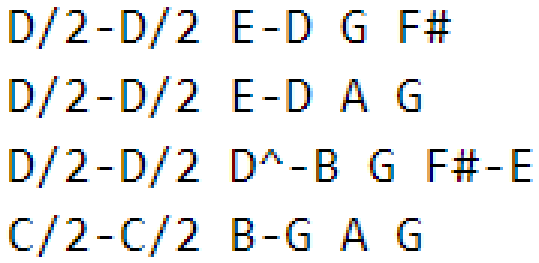


Figure 4.20: Happy Birthday in Saxon notation.

The second way was using the braille line to input the music elements in Saxon notation (Figure 4.20). This would make things harder for us since we would have to translate from Saxon notation to *MusicXML_braille* and then translate it again to *MusicXML*.

Asking the experts

We decided that the best way to solve this question would be to directly ask the professional blind users about it, or, in case we could not reach anyone, people that work with them every day.

Therefore, we proceeded to contact ONCE and ask them which of the two approaches was the best one.

When they came back to us, they told us that the way they write music is the same way *MusicXML_braille* files are written. With that in mind, we proceeded to implement the new functionality

4.4.3 Implementing the functionality

Now we are going to do a little recap on the project parts involved in the development of the new functionality.

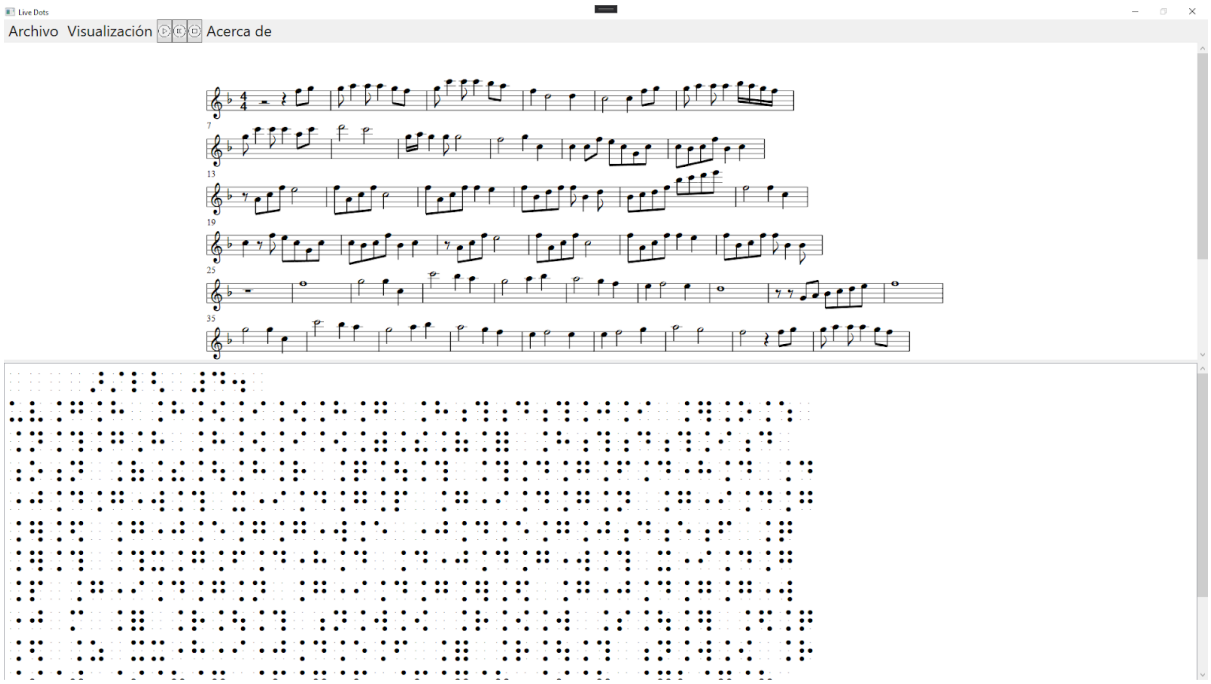


Figure 4.21: *LiveDots window.*

First, it is important to know that the part where the braille score is shown is a *WPF* textbox named text1, which is connected to the *Braille*, *BrailleText*, and *Viewer* attributes of the *MainWindow* class.

Braille is a string conformed by the *ASCII* representation of the braille elements in the score. *BrailleText* is an instance of the class with the same name that has a list of strings with the dot representation of the braille elements in the score and a *Viewer*.

The viewer is an instance of the class *BrailleMusicViewer*, which contains a list of strings with the same name as the class that saves the information needed for the screen viewer *JAWS* to do its work. It also contains two lists of *ints*, which are used because, in dot representation, each music element can occupy more than one dot box. The first list *Forward* saves the information to get from each point of the score to the next music element. The list Backward does the same but gets to the previous music element on the score. These last two are used to update the caret to its correct position between musical elements when navigating the braille score.

Any modification that the user does on the braille score needs to be updated in *Braille*, *BrailleText*, and all the elements of *Viewer*, for the application to properly work. Apart from updating the ink score using *Manufaktura* (Figure 4.21).

The first thing we did was to set the property of *text1.readOnly* to false, so now the textbox would be editable. Now the user could write and delete whatever they wanted on the textbox. However, this only changed the text of text1, but it did not update anything internally, and, as we said before, the three elements explained before needed to be updated in order to work.

The second thing we did was to create two different states of the application, now we would have:

- The navigation state, which was going to be used to move around, listen to the different notes and get used to what is written on the score.
- The editing/modification state is used to write new music elements, change them, or delete them.

Navigation state

This state is the standard state of the application, or to say it in a different way, the only state that the application had before.

In this state, the user can navigate the different functionalities of the application using the "*Tab*" key and once on the braille score, using the arrow keys to move around the score.

As we mentioned before, when navigating around the braille score, the application calculates every time that the caret should move more than one position depending on the length in dot notation that the element on the current position of the caret has. Remember this was calculated using the *Forward* and *Backward* lists.

Editing state

This new state was created to give complete freedom to the user when editing the score because on the navigation state you can only move the caret between complete music elements, while when editing the user might want to move the caret between two braille characters of the same music element.

The application enters this state when the user presses any key that is not the tab or the arrow keys while being on the braille score. Once the application has entered this state, the user can move freely on the score and the application stops calculating the correct position of the caret.

In this way, the user can now write or delete anything they want, and once they finish editing, or want to listen to a small fragment they just created, all they have to do is press Enter to exit edit mode and for all the changes to apply to the whole application.

4.4.4 How does the edit mode work

As said before, when the user presses any input key (not navigation key) the application enters the editing state. When the user wants to finish the editing state, all they have to do is press "*Enter*" key.

Once Enter is pressed, we take the new text that the user has written (*text1* textbox) and create a list of characters from this text. Then we create an auxiliary *BrailleText* and use

the factories to load the instance of the *BXBrailleScore* to an auxiliary variable.

Then, the same way we use the parser when opening a *MusicXML_braille* file, we parse the list of characters and save the parsed result on the auxiliary *BrailleText*.

After that is done all we need to do is update the *MainWindow´s BrailleText* to the auxiliary already parsed one with the new information that the user has inputted. We also update Viewer with the viewer from the auxiliary *BrailleText* and the same with Braille.

Now everything is updated internally in the application part that corresponds to the braille score. Now the only step left is to update the ink score for the sighted users.

To do this we use the parser that parses *MusicXML_braille* to *MusicXML* since we need to send *Manufaktura* a *MusicXML* file to show the ink score. We create the *MusicXML* file from the instance of the *BXBrailleScore* and tell *Manufaktura* to update the ink score.

User input error detection

As the user is given complete freedom to edit the braille score, sometimes they can make modifications that are not valid such as adding music elements after the end of the score mark.

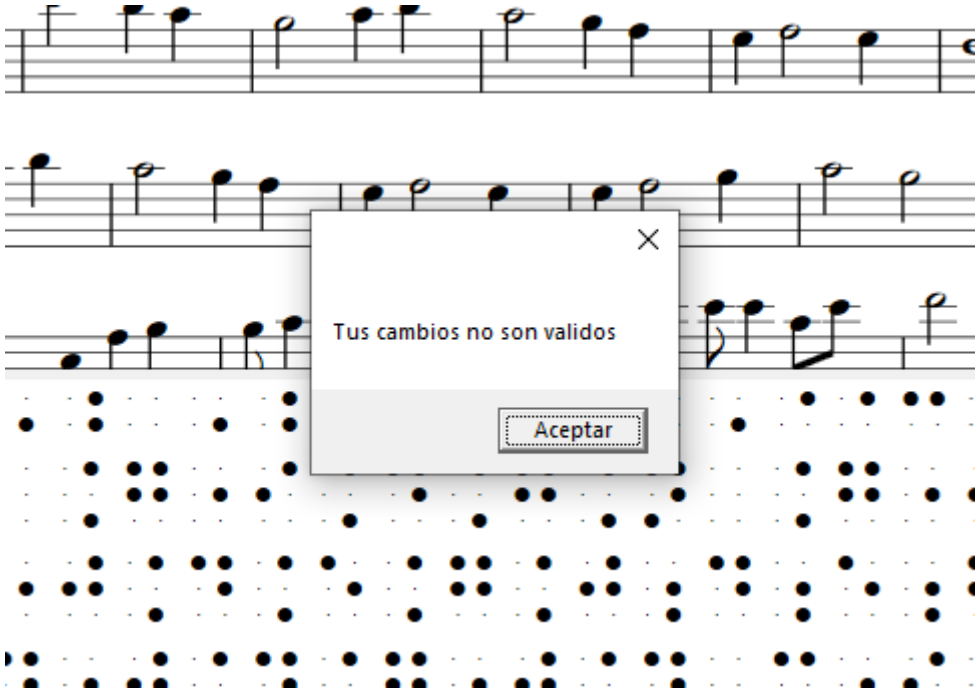


Figure 4.22: Input error pop up window.

When this happens, the application detects it and a pop-up window appears. This window (Figure 4.22) does not allow the user to keep interacting with the application until it has been closed, either with the mouse or pressing Enter. The window keeps appearing until the user fixes the part of the modification that makes no sense.

The way this works is by managing exceptions. When the application is updating all the *MainWindow* elements after the user has finished a small edit, we capture the possible

exceptions caused by the recent user input and throw the error message.

4.5 Improvements based on previous feedback by ONCE

A bug report and feedback document were written for the previous version of the application that we thought was interesting to be noted here, which can be seen in **Annex A** (C).

Many of the feedback written in this document was positive, while also giving possible suggestions and bugs that happened during their testing. Many of the points mentioned are unfortunately out of our reach of knowledge and need the help of an expert in the music field, one example, is that the application was missing some musical elements such as triplets, another example would be that one of the points given commented that when loading more complex musical scores, the MIDI player had issues reproducing notes tied with a prolongation ligature. Some of the issues related to the visual editor were too hard to track as it was a very inconsistent bug and could not be reproduced, and lastly, the screen reader, which involved *JAWS*, these things can be possible future improvements for the application.

One thing that we managed to fix was the braille transcription, were the issue at hand was that the clef sign appeared before the key and time signature in the first line, as it is in an ink score, but according to the Braille musicography rules, it should not be like this, but the clef sign should be written later instead. Now, this is fixed and now it has been properly re-positioned.

4.6 Application code clean up

As we know, this project is a continuation of an application, which was a project last year itself too. After being in two developments, it is natural that the application has a lot of code and elements that need to be removed and cleaned up. So here is a brief summary of the clean up we did towards the final stages of development, the objective of this is to make the code look cleaner and more readable for the future projects that might happen from now on, giving future developers an easier time at the beginning of their development, as it is very challenging to start with something as big as this.

Besides removing the unused "*using*"s that were present in the different scripts, we also removed many variables and enumerators that were not used, these variables usually served their purpose earlier for fast testing but were forgotten and left there. Many commented code was soiling the code in general, so the elements that were considered useless were eliminated.

There were a few classes that served no purpose whatsoever, these were probably created at first but then they were not used and forgotten too, these were removed also. Many classes had a lot of functions and constructors that had 0 references, such as *MainWindow*, where there were more than five or *BrailleMusicViewer*, which had more than ten functions; and all of them were not called by anything, so these were also removed.

Chapter 5

Testing

5.1 Code testing

"Software Testing is a process, which involves, executing of a software program/application and finding all errors or bugs in that program/application so that the result will be a defect-free software. Quality of any software can only be known through means of testing (software testing)." [61].

To create our tests we analyze multiple applications that offer a testing framework and ended the testing capabilities of Visual Studio. Visual Studio allows the creation of the unit, load, and integration test Unit Testing "unit testing tests individual software components or a collection of components. Testers define the input domain for the units in question and ignore the rest of the system. Unit testing sometimes requires the construction of throwaway driver code and stubs and is often performed in a debugger" [62]

Unit testing consists of the creation of tests for each individual unit or component of the code during the development of the application. The objective of this test is to be sure every that class we have implemented worked the way we intended. This test allows us to check methods even if to do it we needed another function to show the result. This means that could debug even if the needed class or method was not implemented yet.

5.2 User Testing

To test how our applications would fare against real users we develop a few scenarios based on how a user would interact.

This test and the procedure we used to realize them can be seen in the product backlog image in the appendix B.

These scenarios can be separated into three, this being:

5.2.1 Score player scenarios

The first task asked the user was to select freely a number of notes and play them to see how the user is capable of utilizing this functionality.

5.2.2 Score modification scenarios

For this test, the user is asked to try to modify the score. This will give us insight into the complexity a user has in modifying the score according to the rules, like maintaining the beat of the score.

5.2.3 Score selection scenarios

This scenery is the most straightforward one. The user is asked to select a score in braille and save it in XML format.

5.3 Test results

Due to Covid Pandemic, we were unable to perform this test with visual impairments individuals. Instead, We received help from David Peña Quineche who did our test acting as a visually impaired user with the help of a refreshable braille display. As it can be seen in the annex C all tests yield good results with a few errors. For the errors reported the first one referring to an error in the playing of the score had been impossible to reproduce, so we could not investigate it further. The second error is an oversight on our part. We assume the user would understand that the same directory from which the original file was selected would be used to save the new score.

Chapter 6

Individual contributions

6.1 Miguel Zhefan Ye Ye

The project started with the proposal of expanding the functionalities of *LiveDots*, which is an application that aids with the inclusion of blind students in music-related classrooms. We were first introduced to the application, and their bachelor's thesis of *LiveDots* [2]; the first objectives were given and I was assigned with the task of making a music player that reproduces the musical notes in the braille music score, this player would reproduce notes according to the position of the user's cursor position, whenever the user moved the cursor, it would play the new note.

The continuation of the application will be on the program itself and it will be done in C# as the programming language.

After this, we proceeded to do an exhaustive study of their application and reading their thesis and encountered some bumps to understand the software properly, as C# was not our most used programming language (in particular, *WPF* projects, which is something we hardly ever used), we took a bit of time to get used to everything that was in front of us. Later, we visited ONCE's tiflo-technology and innovation center where we met some of ONCE's representatives, where they gave us an overview of the project, the impact it has, why it is so interesting, and answered many of the questions we had at the time of the software itself.

Briefly, after this, we set up a Git repository and we each started to implement our parts, the main idea was to start as early as we could with the programming part so we could diminish the risk of not reaching the objectives and to notice if we were over-committing with the objectives that we set.

Before starting to fully commit on the programming side, research needed to be done to understand how Braille worked, since in my background I almost had no interaction with it, my knowledge with it was very limited, after some investigation and meetings with our professors explaining us some topics of Braille, I started to implement my part of the project.

A manual reproduction of a musical note was achieved fairly quickly, as I understood the *Manufaktura* part of the application had already pretty fast, but I encountered many

problems with the variables it asked specifically for the reproduction, creating it manually was easy but it needed to be done automatically.

One of the ONCE's representatives (who also helped us throughout the whole project) gave me a hand to find a way to tackle this obstacle. The solution was found and understood, and by using a *.resx* file, which is an XML essentially, we could convert the musical notes to a format we wanted and use that desired string format. Now we would create a new class that could handle the conversion from a simple string to a variable that *Manufaktura* understands and knows to reproduce in the application (this variable is called Note). This new class handles all the different musical note strings with a dictionary, the dictionary fills its keys and values using an external XML file that tells the dictionary how many octaves we will use and the musical notes of each octave.

This dictionary receives a key, which corresponds to the musical note string and returns a Note value, this is the variable we use in *Manufaktura* right after. So, when the cursor's position was modified, the application would ask the dictionary for the note and this would be reproduced. While we were developing our parts in the software, we would do meetings with our professors (customers) to check the progress and discuss any roadblocks we had. These meetings were handled at the beginning by the customers but later on, they were handled by me.

After successfully completing the task in hand, in the next step, we decided to increase even more the functionality of the note reproduction, by interpreting the multiple selection feature the cursor has, this means that if the blind user were to select more than one note with the cursor, the application would reproduce all the notes that were selected from the beginning to the end. We decided to take the programming route solution for this part, as using musical scores would need the programmer (me in this case) to have knowledge of music.

The solution was to implement asynchronous functions that would wait for the duration of the musical note since C# has an already working architecture that works very well without the need of using threads on your own. Once the application finished playing the current note, it would move to the next one in the list and so on. This way, the application can run while the notes are still being played in the background [63], and if the user decides to deselect or move, the remaining musical notes would not be reproduced.

And lastly, all the meetings with our customers and between us, the developers, were organized by me towards the mid-point in the development. In the beginning, we met up very inconsistently every two to four weeks, so after a bit, we decided to make the meetings more stable and consistent, so tracking every task and progress would be easier, as a fuzzy lane began to appear.

During the rest of the development, we decided to meet up every two weeks, and later on, towards the end of it, we would meet up every week to comment on our progress, discuss the issues we might have gotten and give out tasks for the next week. Meetings, where our customers were present, were also written down and described below in A: Appendix A.

We needed to do a study of what Braille is and the use it has, why was this topic so important and needed. Since this is certainly something that is not mentioned enough. Any student with any type of disability should receive a proper and inclusive education

and more importantly, the removal of the social stigma there is around any person with a disability. We investigated the progress that has been made throughout the years and compared to a few years back, the situation was getting better although there is still a long way to go. This study made us realize how different blind people perceive things and by learning the same things, a blind student would interpret it differently.

Both the writing of the thesis and the development of the project have been a very good experience and an honor to be helping out in the advancement in the use of technologies for the inclusion of blind people. I would like to thank David for his help to make this project take off and be in its best shape possible. This not only has helped us to learn but also to realize how important it is to have an inclusive education. This only leads to me wanting to learn more.

6.2 Álvaro Antón García

This project was a proposal of the ONCE organization to add more functionalities to an already existing musical application, which was capable of reading a score and parse it to a braille score. The settled objectives were to upgrade this functionality by implementing a reverse parse in which a braille score resulted in an XML score. The second functionality we were asked for was to improve the program´s music player.

I was in charge of the reverse parse while my colleagues worked on the improvement of the music player, to allow users to modify the score in the application and playing only the parts selected by them.

The development started with the study of the code and the paper published by the previous team to understand what was already implemented and how to tackle better our tasks.

The code was written in C# utilizing .Net for the framework and WPF and XAML for the visual part.

Following this, I researched how a score works since I had no previous knowledge of music. This research was expanded during the development since new terms and necessities arise.

I started by trying to understand how they managed to pass from and XML to braille. I have to admit that this was harder than it should have been since we received the code before the paper.

An external library was utilized to pass from the XML file to tree pattern. This library called MusicXML.Net [64] allows the creation of a *MusicXML* tree structure without the need to implement new classes and return it in a variable.

With the *MusicXML* tree completed they proceeded to create another tree called BrailleTree which they utilized to obtain the braille score by adding the dots corresponding to the attributes and notes of the ink score.

The BrailleTree consisted of a series of classes forming another tree structure. This tree consisted of a class called *BrailleScore* containing a list of *BraillePart*. In the same regard, this class contains a list of *BrailleMeasure*. *BrailleMeasure* contains the attributes of the score, *BrailleAttribute*, and a list of *BrailleStaff*. *BrailleStaff* indicated with which hand

the notes are played and another list of *BrailleVoices*. Finally, *BrailleVoices* contains a list of *BrailleNote*.*BrailleNote* represents the notes of the score containing all the information needed. Since my part did not rely so much on the visual parts of the application I focused on understanding the basic concepts. For this, I only needed a list containing the braille and a score in XML format that the library *Manufaktura* was capable of reading. The last part was the conversion of the braille score to a format capable of being saved to a file. For this, they created a new class that contains a dictionary giving each braille sign a char value. With the understanding of how the code works I started with the development of my part.

The most logical way to implement the parser was to imitate the one already implemented but backwards, reading the MusicBraille file, filling the *BrailleTree*, the *MusicXML* tree, and creating an XML file.

The process of reading the file was done in the first place just reading the file char by char and using a string, but during the process to pass from a char to the braille score I ran into the problem of not knowing what characters correspond to what part of the score. To solve this problem, the same process was applied but using a list of characters that let me remove the necessary number of elements in the list, in accordance with the number of characters needed by the attribute or the note. For the *BrailleTree* the structure was similar to the one already implemented to the point that in the early development the same classes were used, this resulted in a hard-to-understand code. To solve this the classes were separated and implemented using a factory method so as only one instance of the method was used.

The *BrailleTree* was completed by going over the list of characters and using cases to give the object its corresponding value and adding the dots that correspond to the character to a list of strings.

With the *BrailleTree* complete it was now time to create the XML tree but going back to the study of the code the necessity of this tree was deemed unnecessary since this was an extra step that accomplished nothing for this functionality. The XML file with the ink score can be achieved with the information found in *BrailleTree*. The creation of the XML file was a simple but tedious process of going class by class of the *BrailleTree* following its structure and adding the tags with the information. The last part to implement was the visualization of the ink and braille score and the ability to save the XML.

For the visualization, it was only a matter of understanding the class Viewer created by the previous team and knowing what elements were needed to change to visualize the scores. To do this we needed an XML that the library *Manufaktura* was capable of reading, to show the ink score and the list of strings containing the braille dots. To save the XML the library *MusicXMLParser* was used to obtain a string with the XML I created before that was changed in the class Viewer and using the internal library System.IO.File save it with the same name as the braille file and changing the encoded form. *MusicXML_braille* to *MusicXML*

6.3 Álvar Julián de Diego López

When we were presented with the project by Maria and Joaquín, who explained to us that it was a development project about a music editor in braille for the ONCE organization,

we all loved the idea and were excited about the project.

The first thing that I and my mates had to do was to get to know the project and study all the code deeply so we would be able to get hands down on it and start developing new features and functionalities for it.

While studying the project I encountered that there were many things that I did not fully understand because I had barely any knowledge in music, so for a brief time I switched my studies from the previous project code to learn music basics. After I learned some of the basics about music, I went back to the project and everything was much easier to understand. I had to do some recap on C# since we had barely worked with it for some years.

The first tasks we were assigned were to create a parser that would parse from *MusicXML_braille* to *MusicXML* and to create a system that would allow the user to reproduce the notes on the score by navigating on it with the navigation keys. Álvaro started working on the parser while Miguel and I started working on the reproduction of the notes.

The original project allowed the user to do slight modifications to the ink score by dragging the notes with the mouse, which worked because *Manufaktura* gave that option. But if the user changed something and then tried to reproduce the score, it would not reproduce the changes. Taking this into account, I decided to start working on it since it would be needed for the future when we worked around the reproduction of the braille score. I came up with a way of fixing this, which was to renew the ink score when it is edited so that the Player would now reproduce the changes also.

After doing that, Miguel and I decided to ask María and Joaquín for more tasks because we thought that the reproduction of the braille score was enough work for one person. I then got assigned the functionality that would allow the users to edit the music score directly in braille.

First, I started looking for other music editors to see how they worked around this functionality, but I quickly realized that it was not going to be as easy to implement in braille as other non-braille editors had it. We contacted ONCE to ask them what they thought and they gave us the idea to implement it as if it was a regular text editor and not a music editor.

Knowing that I got to work and developed this functionality. I had to work closely with Álvaro because I ended up using the parser he made for my implementation which was very convenient. And with Miguel, because we had to figure out how to consolidate the editing with the reproduction of the music at the same time.

This project has been a really good experience because I have learned how to continue working on a project that was started by other people and that will be continued also by different people. I really appreciate that we have had the opportunity to learn how accessible applications are developed and how important they are, not only in music but in every aspect of life.

I want to thank Miguel and Álvaro for being such good mates, David for all the help he has given us, Maria and Joaquín for being excellent directors, and ONCE for allowing this project to exist.

Chapter 7

Possible improvements for the future

This project was done to improve the already existing application and expand its functionalities. Allowing the users to be able to have multiple options, maximizing the interactivity is the priority. As for possible improvements, the application can be improved in many other ways as well as increasing its interactivity:

- **Increase the number of beats allowed that can be reproduced with the cursor.** As of now, 4 types of beats are implemented and can be reproduced freely using the cursor, moving around the braille, or selecting a group of braille text. There are more types of beats that are less conventional but are sometimes used. This would increase the number of musical scores the application can reproduce.
- **Implement the option to export from the Braille form to a *png*, *jpg*, or *MusicXML* format, showing the actual musical score.** This allows the user to be able to add the notes to the Braille and then save up the changes the user has done. Allowing them to personalize the musical score and even opening up the possibility of creating a complete musical score of their own from zero.
- **Increase the functionalities of the sound and the cursor in the braille musical score.** Currently, the cursor can select when it is dragged upside down, this can be improved by allowing the user to also drag from left to right or vice versa. Also assigning a special key to reproduce the sound rather than sounding automatically could be very useful, this is also applied to the group of notes.
- **More complexity to the score.** As of now the score the program can produce is quite limited with only being able to use simple notes. In music, the array of elements a score can have is quite massive with elements like partial repetition, key signature of a note, or ties changing completely the way the score is played.
- **Threads.** During the study of the code we came across multiple problems like the application closing due to spending too much time debugging and needing a restart of the computer to be able to relaunch the application. This was caused by the use of threads without the implementation of a mechanism in case something went wrong.
- **Variety of instruments.** The application can only use piano sounds to reproduce

the score, a possible extension for the application would be to allow the user to choose the desired instrument to reproduce the selected score. Allow the blind user to know what instrument they are selecting by saying the instrument's name with voice.

Chapter 8

Conclusions

LiveDots was born from the necessity of having more inclusive applications for blind people, such as EDICO, which was created for maths, physics, and chemistry by ONCE; LiveDots arose for musical purposes. Since the first development cycle was successful and it was well-received, ONCE decided to carry on with the project, we had the opportunity to be the ones in charge of the continuation of the development, expanding the functionalities of the application.

Our project repository can be found in the following link: `https://github.com/alvant01/TFGLiveDots`, here you can see all the progress we have made since the start of the project until the very end of the development.

During the development of this project, we ran across two challenges. The first was the lack of knowledge of music. In the very beginning, each of us had limited knowledge of how a score works, different types of notes, etc; moreover, we needed to tweak some elements in the application to make them work for our objectives. This led us to research the complexity of score creation to expand our knowledge on this matter.

The second and most important was the development of assistive technology. Although during our degree we had developed multiple applications, the use of assistive technology to make them inclusive was lacking. This meant that every design we have implemented to interact with the user should be thought of as having a visually impaired user in mind. Therefore, we not only had to implement shortcuts but also, to learn how music is composed in braille. So, this has been an eye-opening experience, learning how braille language is applied to music is an exceptional opportunity.

During the research of the state of the art, we realize the limited options people with visual impairments have when it comes to music applications. This made us realize the few opportunities we programmers give to people with disabilities and society in general. It is easy to make applications based on how we would use it, but putting yourself in others' shoes it's the hard part. Most of the time due to the increment in cost and time making a program accessible we block potential opportunities for many users that could make use of it.

When we started working on improving *LiveDots*, the application was capable of opening *MusicXML* files, and showing them in ink and braille formats. It had a simple music player which could reproduce simple scores from start to finish and it could save the

files into *MusicXML_Braille* format. It was functional with the screen reader JAWS and everything could be used with the braille line.

After our development cycle has finished, *LiveDots* now has a new big amount of functionalities that make the application more professional and useful for visually impaired users. *LiveDots* can now represent *MusicXML* and *MusicXML_Braille* files in ink and braille formats and it can translate from *MusicXML* to *MusicXML_Braille* and vice versa. It now has complete reproduction control over the musical score, giving the option to reproduce the whole score, a selected part or not by note. Last but not least, *LiveDots* now has a functionality that allows the users to completely modify the braille score in real time, which makes the application be at the same level than other music notation software that is not accessible.

All these new functionalities work as intended and were tested by a member of ONCE who gave us a feedback report that can be seen on **Annex B** (C). In this report it is stated that the objectives that were set for this project have been achieved. It also states there are minor problems which involve complex music notation, and gives some advice on how to fix them and improve the application in the future.

For this, we are proud of having been able to continue the development of *LiveDots* and for the opportunity granted to us by ONCE. Moreover, we hope that the team in charge of the next development cycle has a delightful experience and learns as much as we had. To end with, we hope and believe that this project will grant an accessible tool to develop one of the finest forms of art humanity has created, music.

Bibliography

[1] Redacción, "España tiene el porcentaje más alto de personas con discapacidad visual en Europa," 2021.

[2] Ó. D. Ribagorda, C. G. García-Heras, and L. d. T. Barrio, *Development of a tiflo-application for translating music scores to braille.* PhD thesis, Complutense de Madrid, 2020.

[3] Wikipedia, "Musical notation." `https://en.wikipedia.org/wiki/Musical_notation`, 2003.

[4] vitaltech.org., "Humanware Brailliant." `https://www.vitaltech.org.uk/theme/refreshable-braille-displays-and-tablets/`.

[5] R. Barczyk and D. Jasińska-Choromańska, "Experimental studies of the quality of embossed characters of the Braille alphabet," *ResearchGate*, 2016.

[6] H. L. Cooper, "Una Breve Historia de los Sistemas de Escritura Táctil para Lectores con Ceguera e Discapacidades Visuales." `https://www.tsbvi.edu/seehear/spring06/history-span.htm`, 2010.

[7] Wikipedia, "Moon type." `https://en.wikipedia.org/wiki/Moon_type`.

[8] M. Buja and T. Mun, "FONTES ARTIS MUSICAE," *Journal of the International Association of Music Libraries, Archives and DocumentationCentres (IAML)*, vol. 57/2, 2010.

[9] D. Goto, T. Gotoh, and T. Reiko, "A Transcription System from MusicXML Format to Braille Music Notation," 2007.

[10] Upv, "Síntesis del sonido El protocolo y el formato MIDI." `http://www.disca.upv.es/adomenec/IASPA/tema5/Midi.html`.

[11] ASSOCIATION GIUSEPPE PACCINI, "The new Braille Music21." `https://braillemusiceditor.com/`, 1996.

[12] Dancing Dots, "GOODFEEL."

[13] Dancing Dots, "The Lime Lighter Music-Reading Solution for People with Low Vision." `https://www.dancingdots.com/limelighter/limelightermain.htm`.

[14] MakeMusic, "Finale: Music Notation Software That Lets You Create Your Way."

[15] Avid Technology, "Music Notation Software - Sibelius." `https://www.avid.com/es/sibelius`, 2019.

[16] J. Nieuwenhuizen, H.-W. Nienhuys, W. Lemberg, J. Warchoł, C. Sorensen, and D. Kastrup, "LilyPond: Music Notation For Everyone." `lilypond.org`.

[17] Wikipedia, "Diagram of treble, alto and bass clefs with identical-sounding musical notes aligned vertically." `https://en.wikipedia.org/wiki/Clef`, 2003.

[18] The Shodor Education Foundation, Inc., "Basic Signs." `http://www.brl.org/music/manual/basic/index.html`.

[19] Music Theory, "Octave Registers and the Piano." `https://www.allaboutmusictheory.com/piano-keyboard/octave-registers/`, 2013.

[20] A. H. Gregory, "The roles of music in society: The ethnomusicological perspective," 1997.

[21] F. Kersten, "Music as Therapy for the Visually Impaired," *Music Educators Journal*, vol. 67, no. 7, pp. 63–65, 1981.

[22] Ian D. Bent, "Musical notation." `https://www.britannica.com/art/musical-notation`, 2019.

[23] B. Benward and M. Saker, *Music in theory and practice*. 8.a ed., 2009.

[24] Wikipedia, "Pythagorean tuning." `https://en.wikipedia.org/wiki/Pythagorean_tuning`.

[25] R. de Candé, *Nuevo diccionario de la musica / New dictionary of music*. Ma Non Troppo, ilustrada ed., 2002.

[26] PharmaBraille, "What is Braille." `https://www.pharmabraille.com/pharmaceutical-braille/what-is-braille/`.

[27] J. M. Abramo and A. E. Pierce, "An Ethnographic Case Study of Music Learning at a School for the Blind," *JSTOR*, vol. 195, p. 16, 2013.

[28] E. Murphy, P. Mccarthy, M. Shevlin, and A. Heelan, "Giving voice to blind and visually impaired students transition experiences, addressing gaps in policy provision," tech. rep., AHEAD, Trinity College, Dublin, 2015.

[29] F. W. Moss, *Quality of Experience in Mainstreaming and Full Inclusion of Blind and Visually Impaired High School Instrumental Music Students*. PhD thesis, University of Michigan, 2009.

[30] W. Homenda, *Breaking Accessibility Barriers: Computational Intelligence in Music Processing for Blind People*. Warszawa: Springer, Berlin, Heidelberg, 2008.

[31] A. Capozzi, R. D. Prisco, R. D. Prisco, and R. Zaccagnino, "Musica Parlata: a methodology to teach music to blind people," 2012.

[32] B. Encelle, N. Jessel, J. Mothe, B. Ralalason, and J. Asensio, *BMML: Braille Music Markup Language*. 2009.

[33] ONCE, "Braitico." `https://educacion.once.es/braitico`.

[34] J. M. Carenas, A. B. Cabra, M. G. Mata-García, P. C. Gea, and D. H. Hernández, *Prácticas Edico ¿Qué es Edico?* 2018.

[35] wcblind.org, "What is Refreshable Braille Anyway?." https://wcblind.org/2018/02/what-is-refreshable-braille-anyway/.

[36] freedomscientific, "JAWS." https://www.freedomscientific.com/products/software/jaws/.

[37] NVAcess, "NVAcess." https://www.nvaccess.org/.

[38] Perkings.org, "screen-magnification." https://www.nvaccess.org/https://www.perkinselearning.org/technology/screen-magnification.

[39] Encyclopaedia Britannica, "Score." https://www.britannica.com/art/score-music, 2017.

[40] E. B. Bordonau, "DE LA TRADICIÓN ORAL A LA ESCRITA ENTRE LOS MÚSICOS CIEGOS ESPAÑOLES: LOS SISTEMAS MUSICOGRÁFICOS DE GABRIEL ABREU Y PEDRO LLORENS," *JSTOR*, vol. 32, pp. 151–163, 2009.

[41] B. Krolick, "New International Manual of Braille Music Notation." http://www.brl.org/music/index.html, 1998.

[42] Wikipedia, "MusicXML." https://en.wikipedia.org/wiki/MusicXML.

[43] Fundación Juan March, "¿Por qué MusicXML y no MIDI en el contexto de las bibliotecas?." https://www.march.es/bibliotecas/tme/musicologia/musicxml.

[44] G. Paccini, "Braille Music Editor." https://braillemusiceditor.com/about/.

[45] Dancing Dots, "No Title." https://www.dancingdots.com/main/index.htm, 1992.

[46] Dancing Dots, "GOODFEEL® Braille Music Translator." https://www.dancingdots.com/main/goodfeel.htm.

[47] Dancing Dots, "What is Lime?." https://www.dancingdots.com/prodesc/lime.htm.

[48] M. Lang, J. Lung, A. Repain, A. Marzin, C. Sacc, J. Royer, M. Lang, M. S. Kainz, N. Froment, and X. Louba-tier, "Freedots," 2009.

[49] Google Code, "Goodle code archive: freedots." https://code.google.com/archive/p/freedots/.

[50] N. Froment, "MusicXML to Braille converter." https://musicxml2braille.appspot.com/.

[51] Wrike, "What Is Agile Methodology in Project Management?."

[52] M. Beedle, K. Beck, A. V. Bennekum, A. Cockburn, W. Cunningham, F. Martin, and D. Thomas, "Manifesto for Agile Software Development," 2001.

[53] J. Spolsky, "Trello," 2011.

[54] Microsoft, "Microsoft Windows Presentation Foundation." http://msdn.microsoft.com/en-us/library/ms754130.aspx, 2011.

[55] WikiBooks, *C Sharp Programming*. WikiBooks, 2008.

[56] C. Foundation and OpenSeadragon, "Manufaktura Controls." http://manufaktura-controls.com/.

[57] Microsoft Corporation, "xmlserializer." https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer?view=net-5.0.

[58] Musiclever, "Seven notes," 2021.

[59] Microsoft, "ASP.NET Web Page Resources Overview," 2014.

[60] D. Pine, N. Schonning, B. Wagner, D. Lee, N. Turn, M. Wenzel, A. A, M. Jones, M. Hoffman, and L. Latham, "How to: Cancel a Task and Its Children," 2017.

[61] K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pp. 77–81, 2017.

[62] J. Whittaker, "What is software testing? and why is it so hard?," *IEEE Software*, vol. 17, no. 1, pp. 70–79, 2000.

[63] P. Carter, "Async en profundidad," 2016.

[64] V. DARON, "Musicxml.net." https://github.com/vdaron/MusicXML.Net, 2013.

[65] E. B. Bordonau, "EL SISTEMA MUSICOGRÁFICO DE GABRIEL ABREU Y SU APLICACIÓN EN LA ENSEÑANZA MUSICAL PARA CIEGOS EN ESPAÑA (1854-1950)," *JSTOR*, vol. 27, pp. 1099–1113, 2004.

Part A

Appendix: Meeting records

First session. In the first session, we received an explanation regardest the state of the program, what was done by the previous team, and the objectives set for this project. With these objectives in mind, we proceeded to analyze the cost of implementation and divided it according to our assessment. Alvaro Anton was in charge of the parse from braille to XML since this process was the inverse of what was already implemented. Miguel Zhefan and Alvar Julian took the responsibility of upgrading the already existing music player by allowing the user to reproduce a selected part of the score and the implementation of new functionality to modify the score. In this session, we were also introduced to David, a master's degree student who will help us during the development and give us the original code. Finally, we establish an estimated date to have the code done and a revision every two weeks to show our progress.

Second session. The time from the first session to the second was used to familiarize us with the code with the second session being used to answer our questions about it. Additionally, we decided to utilize GitHub as our control version tool.

Third session. Regular session in which we showed the progress made since the last session and received feedback.

Fourth session. In this session, in conjunction with showing our progress and receiving feedback, we agreed to show a functional prototype for the next session. This prototype was to be able to reproduce a note selected with the cursor and be able to read a braille score, parse it, and show a visual representation of the braille score and the ink representation

Sessions 5th to 7th. These sessions were quite similar, every two weeks we will maintain a reunion with our tutors, show our progress and set new goals for the sessions. There are a couple of notable pieces of feedback we received during these sessions. Restructuration of all the code to implement factories with the idea of clarifying the code for future implementations. Change of the object class Viewer to allow more classes to implement it.

Visit the ONCE organization. Although the nature of these projects meant that multiple visits to ONCE´s building were needed to familiarize ourselves with not only the musical braille terminology but also with the electronics components utilized by visually impaired people to interact with our application. Sadly due to the circumstances generated by the COVID pandemic our visits were restricted to only one. During this visit, we were able to speak with David and Pablo who were working on a similar project oriented to the field of mathematics. In addition, we were able to share our ideas and doubts with them allowing us to grasp a better understanding of the future development of our code.

Eighth session. Another regular session without a notable occurrence. We updated our progress and set the date to show our code running. This marks the start of the creation of this document

Part B

Appendix: Product Backlog

Identificador (ID) de la historia	Enunciado de la historia			Priorización	Criterios de aceptación				
	Rol	Característica / Funcionalidad	Razón / Resultado		Número (#) de escenario	Criterio de aceptación (Título)	Contexto	Evento	Resultado / Comportamiento esperado
E1	Como usuario de la aplicación	Necesito poder reproducir notas individuales	Con la finalidad de poder escuchar fragmentos de la partitura	Must	1	Mover el cursor por la partitura braille	Movimiento general	El cursor se mueve en la dirección indicada por el usuario	Debe de reproducirse cada nota del fragmento independiente.
					2	Mover el cursor por la partitura braille	Movimiento dentro de un lugar inadecuado (silencio, clave de sol, etc)	El cursor se mueve en la dirección indicada por el usuario	No se reproduce la simbología
					3	Mover el cursor por la partitura braille	Movimiento a gran velocidad	El cursor se mueve en la dirección indicada por el usuario	Debe de reproducirse cada nota del fragmento independiente.
E2	Como usuario de la aplicación	Necesito poder reproducir grupales	Con la finalidad de poder escuchar grupos de fragmentos de la partitura	Must	1	Seleccionar con el cursor por la partitura braille	Selección general	Queda marcado en azul el fragmento seleccionado	Debe reproducirse cada nota de la selección en el orden correcto e independientes entre sí
					2	Seleccionar con el cursor por la partitura braille	Selección general con elementos que no son reproducibles	Queda marcado en azul el fragmento seleccionado	Debe reproducirse cada nota de la selección en el orden correcto e independientes entre sí, la simbología no se reproduce
					3	Seleccionar con el cursor por la partitura braille	Selección general en distintas direcciones y velocidades	Queda marcado en azul el fragmento seleccionado	Debe reproducirse cada nota de la selección en el orden correcto e independientes entre sí, la simbología no se reproduce. Siempre de arriba abajo, izquierda a derecha
E3	Como usuario de la aplición	Necesito poder editar la partitura braille	Con la finalidad de componer música	Must	1	Borrar elementos musicales	Acortar la partitura	Se pulsa la tecla de eliminar	Debe eliminarse el elemento musical que se encuentre en la posición del cursor
					2	Añadir elementos musicales	Alargar la partitura	Se pulsa cualquier tecla que no sea de navegación o eliminar	Debe añadirse el elemento correspondiente a la tecla pulsada en la posición del cursor
					3	Cambiar elementos musicales	Modificar la partitura sin que cambie de tamaño	Se pulsa la tecla de eliminar y a continuación cualquier tecla que añada elementos muscales	Debe eliminarse el elemento de la posición del cursor y ser sustituido por el correspondiente a la tecla pulsada
E4	Como usuario de la aplicación	Apertura de archivo	Con la finalidad de abrir una partitura en braille	Must	1	Selecionar una partitura en braille	Apertura general	Se selecciona un archivo xmlMusic_Braille	Se muestra la partitura correcta en tinta y en braille
			Con la finalidad de guardar la partitura en formato XML	Must	2	Se guarda la partitura con un formato con el que se pueda volver a abrir con el programa	Guardado general	Con una partitura abierta se da a la opcion de guardar	El participante no encuentra donde se guardo la partitura

Figure 8.1: Product Backlog

Part C

Appendix: Detailed class diagram

The backbone of our application is at *MainWindow*, where all the core elements are located at. Depending on the event type the user inputs, it may be processed outside this class. For example, Braille is initialized in its factory and later its data structure will be used depending on if we are trying to load an XML file or convert Braille back to XML, depending on which case it is, a different tree structure will be used, this tree structures can be seen in Figure 8.2, which corresponds with the conversion from Braille back to *MusicXML*, and Figure 8.4, which shows a similar structure as Figure 8.2, but this time it translates *MusicXML* to musical Braille.

Commands such as play, pause, and stop are handled by *LiveDotsCommands*'s *PlayerCommand*, as shown in Figure 8.3. *LiveDotsCommands* also handles all the events related to file managing, if the user wants to save the Braille score or a *MusicXML* file, open a *MusicXML* file or increase the sizes of any of the mentioned scores above. *LiveDotsCommands* has all the functions it needs to handle all those events, as shown in Figure 8.3. If any of the events are related to open/save files, *LiveDotsCommands* uses a class called *MusicXMLBrailleParser* if the user is trying to save a Braille file and (Converter) *MusicXMLParser* if the user is trying to open a musical score.

MusicXMLParser uses a data structure which is also shown in Figure 8.3 to store all the values that are read from the file. *MainWindow* handles all the events related to changes in the cursor position, this position is then sent to another class called *BrailleMusicViewer*, where it interprets the cursor so it returns the correct position in the braille score according to the position of the ink score, in case, to refresh the mind, the musical notes in the Braille score do not align with the musical scores in the ink score, as musical notes may take several Braille boxes to be represented, so a remapping needs to be done so the musical notes in the Braille score are aligned with the ink score. Obviously, this class also contains the actual Braille score, which is aided by a *DicBraille* to get the translations and *BrailleText* that sets the Braille score properly in the screen so it fits in its assigned place.

In the sound department, the cursor sound is handled by *CursorPosSound*, this class uses a dictionary of pitches that are read from an external XML and uses another *.resx* file to convert the cursor's pointing string to a playable musical note, the conversion is done with the functions in the class *StringToNote*. All the sounds are done in the class *MyMidiTaskScorePlayer*, where all the functions related to *Manufaktura* are located, this class inherits the functions we will use from it. If the user wants to play a musical score (by hitting the play button), is playing single musical notes with the cursor, or selects a group of musical notes, they will be sent to *MyMidiTaskScorePlayer* to be played.

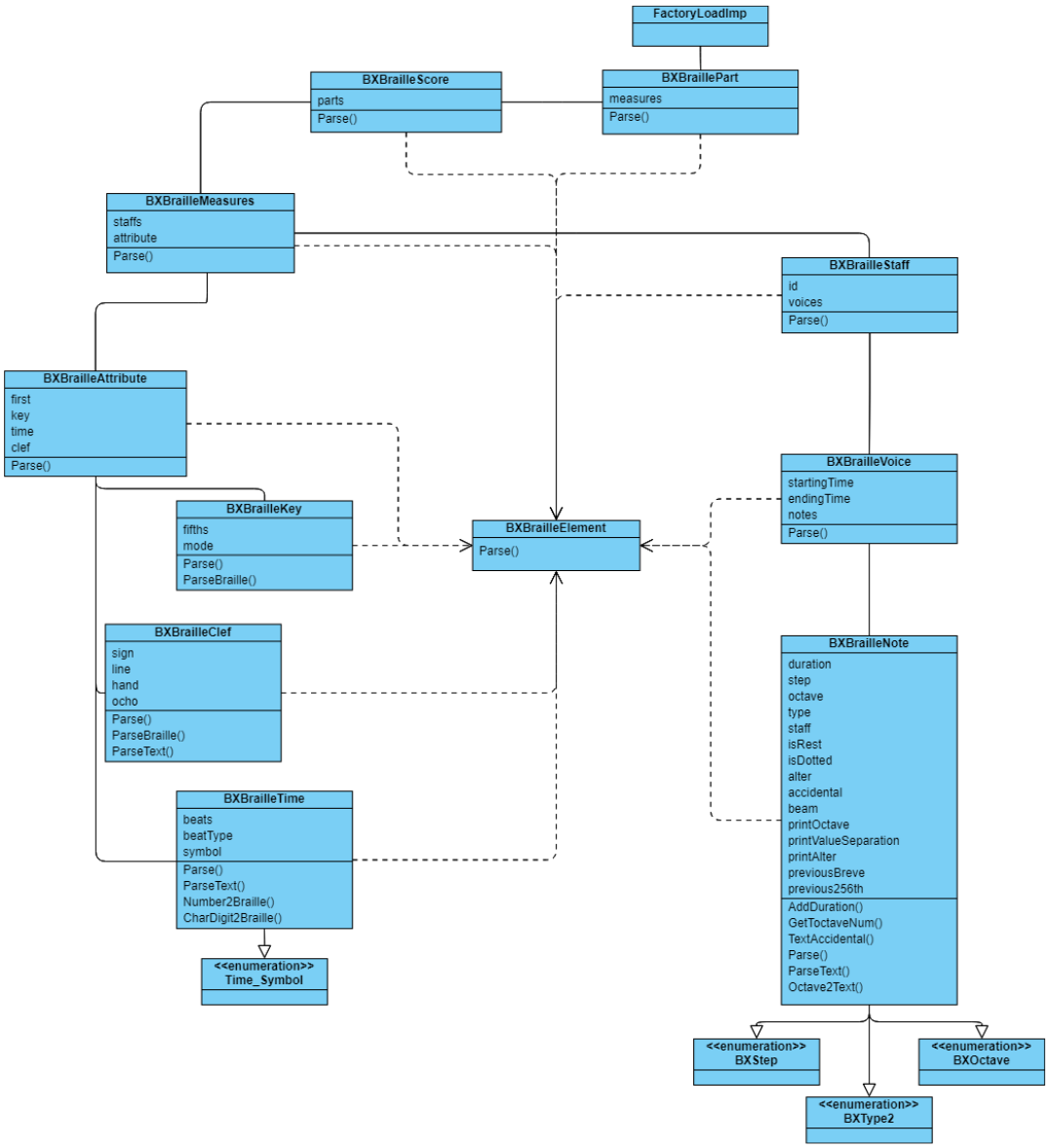


Figure 8.2: Braille to *MusicXML* tree translator

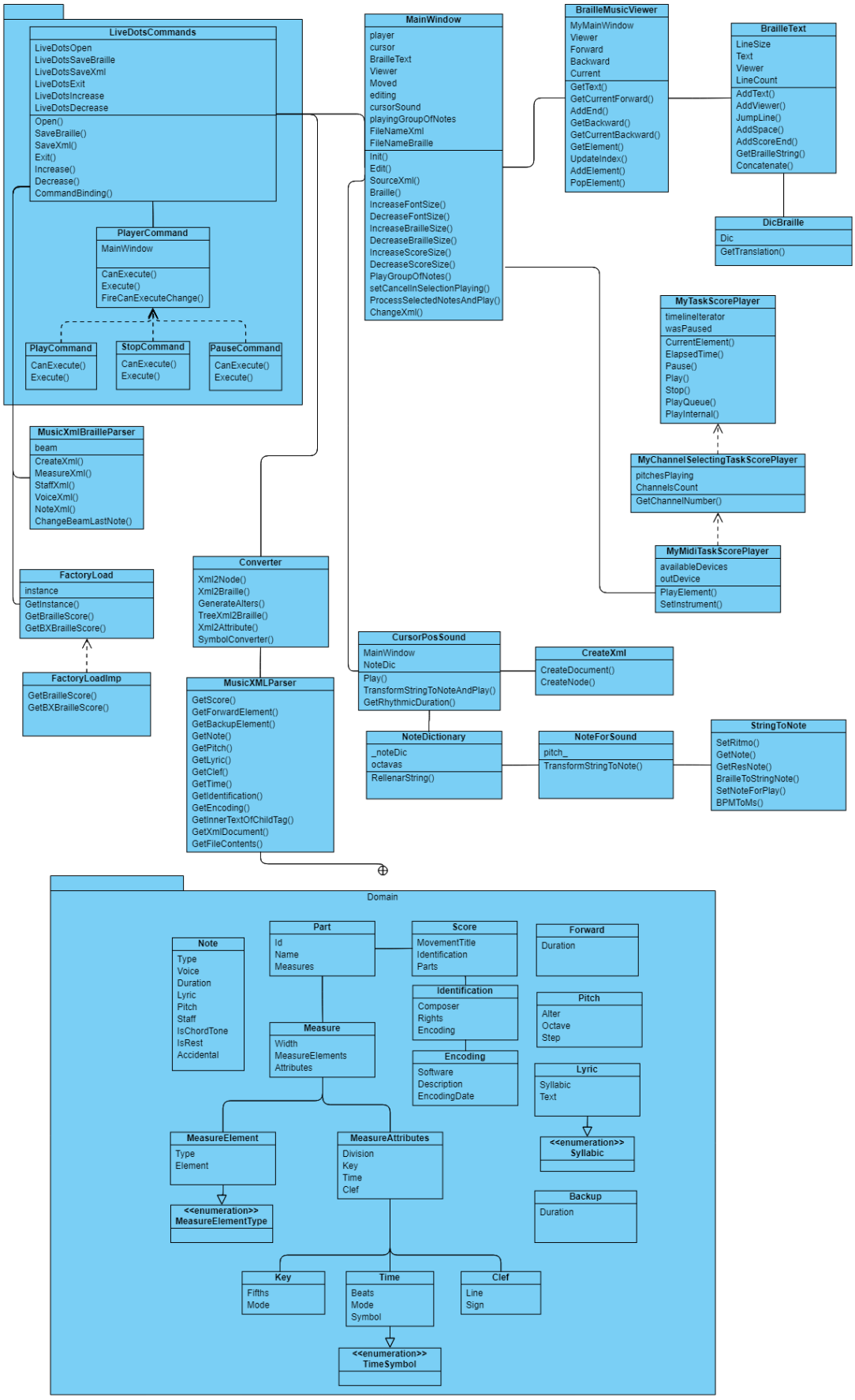


Figure 8.3: *LiveDots* Main software structure

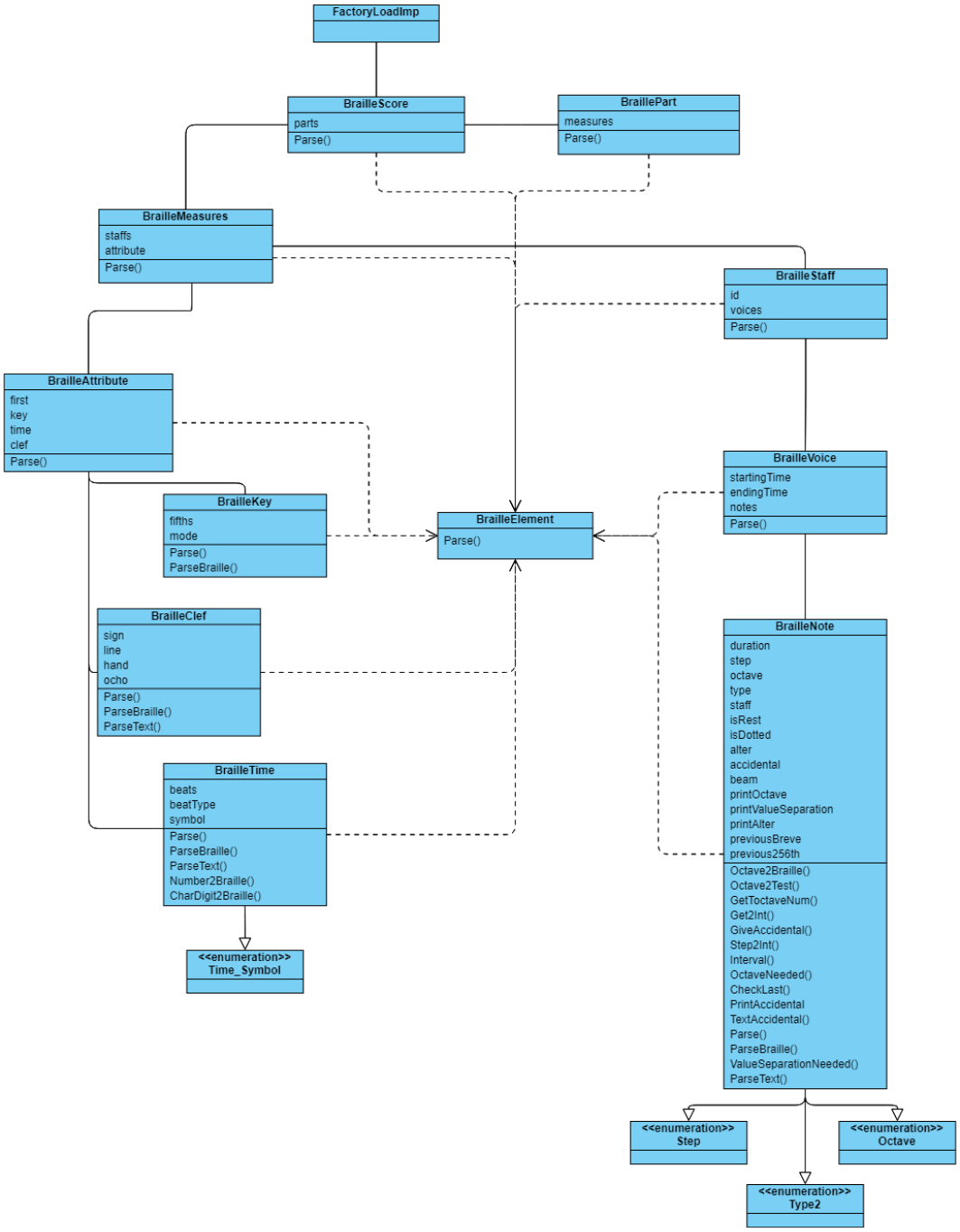


Figure 8.4: *MusicXML* to Braille translator

Part A

Annex: Software feedback report

06/04/2021

Pruebas con el programa de conversión MusicXML a Braille “Livedots”

Primeras impresiones:

En primer lugar cabe decir que todo esfuerzo que se dedique a la realización de materiales, software, etc, destinados a fomentar la alfabetización musical, son muy bienvenidos, y como especialistas de música valoramos muy positivamente esta iniciativa.

A continuación detallamos nuestras observaciones, después de probar Livedots con archivos XML y archivos MusicXML generados con los editores musicales visuales “Finale 25” y MuseScore, en el entorno de Windows 10 y usando Jaws 2020. Comentamos de paso que no se han podido abrir archivos XML generados con el editor de partituras Braille BME V2.

1. REPRODUCCIÓN MIDI: Una vez abierto un archivo correspondiente a una partitura con una sola voz, observamos que la reproducción midi se escucha correctamente, en general, incluyendo los fragmentos entre signos de repetición (aunque no interpreta los signos de repetición cuando solo se hallan al final de la partitura o un fragmento). De vez en cuando se escucha algún error de reproducción en la duración de alguna nota.
   Cuando se trata de partituras más complejas, con dos o más voces, o partituras de piano, el reproductor midi presenta más limitaciones, produciéndose más errores en la reproducción.
   No reproduce correctamente las notas ligadas con ligadura de prolongación.

2. EDITOR VISUAL: Las partituras visuales a una voz se abren correctamente, aunque eventualmente se aprecian problemas con los gráficos de las ligaduras de expresión y no aparece ningún matiz de la partitura original (notas picadas, etc.).
   Hemos observado que algunas partituras a dos o más voces se abren correctamente, pero otras no, no habiendo encontrado todavía una explicación a esta cuestión. Las partituras para piano no se visualizan correctamente, o solo se visualizan parcialmente.

3. TRANSCRIPCIÓN BRAILLE: En primer lugar observamos que el signo correspondiente a la clave aparece en la primera línea, antes de la armadura y el compás (tal y como es en tinta), y según establece la normativa de la musicografía Braille, este signo se escribe después, en la segunda línea, antes de la primera nota o signos que la precedan. Es correcta la transcripción de esa primera línea con la armadura y el compás con una sangría de cuatro espacios.
   Los signos de octava aparecen correctamente en todas las pruebas realizadas, según la normativa específica para estos signos. También los signos de armadura, compás, alteraciones y notas, aunque echamos en falta los puntillos, los tresillos y otros grupos irregulares, los signos de repetición, las ligaduras de prolongación, signo de calderón, todos imprescindibles en una partitura de nivel elemental, así como toda la signografía referida a dinámicas y matices.

Part B

Annex: David Peña's Testing report

Aspectos generales

Comentar un poco los escenarios de las historias de usuario del product backlog y comentar la prioridad en la que se han ido implementando y modificando para que estuviera lista una primera versión funcional para el TFG.

Escenario 1 (E1)
La reproducción de notas individuales en los 3 casos es correcta. Reproduce las notas una a una aunque no compongan un fragmento armónico de la partitura.

Error: cuando se quiere volver a reproducir la partitura completa, se escucha como si estuviera mezclado con las notas que se habían seleccionado antes.

Como futuro trabajo añadiría lo de mejorar esto para que se puedan reproducir fragmentos enteros en función de las características rítmicas de la melodía.

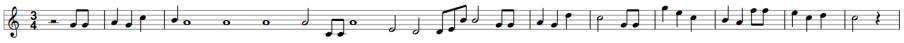
Escenario 2(E2)

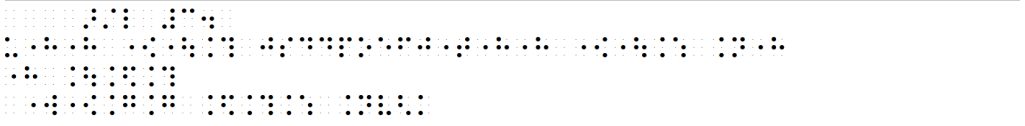
Se reproducen en orden correctamente.
Se mantiene el error del E1.

Escenario 3 (E3)

Al escribir algo erróneo aunque no respete el ritmo y la armonía de la partitura, se añaden correctamente y de la misma manera se reproducen correctamente tanto como notas individuales como la melodía completa.

Se han añadido notas de distintas duraciones con respecto a la partitura de “Feliz Cumpleaños” y ha funcionado correctamente.





Escenario 4 (E4)

Parsea correctamente la partitura de braille a una visual en el editor.

Error: No encuentro ruta donde se guardan las exportaciones o no las realiza correctamente.

Como futuro trabajo añadiría lo de indicar por voz y un MessageBox o algo parecido la dirección de la ruta donde se guardan las exportaciones.

Por lo general, las funcionalidades principales que debe de tener vuestra extensión, cumple con los objetivos pedidos para el trabajo. Además, es compatible con la linea braille. Solo mencionar que es accesible en cuanto a los botones de las herramientas, pero no se puede mover el foco al editor, eso es un problema ya que le quita accesibilidad para los discapacitados en visión.